TDDC03 Projects, Spring 2007

Gömda kanaler – hemlig dataöverföring i en simulerad MLS-miljö i operativsystemet FreeBSD 6.2

Jerry Falkcrona
Jonas Tevemark

Supervisor: Viiveke Fåk

Gömda kanaler - hemlig dataöverföring i en simulerad MLS-miljö i operativsystemet FreeBSD 6.2

Jonas Tevemark Jerry Falkcrona
Linköpings universitet, Sweden
Email: {jonte466, jerer606}@student.liu.se

Sammanfattning

Att designa ett MLS-system så att vissa användare inte kan kommunicera med varanadra är ingen lätt uppgift. Att begränsa sockets, IPC-kanaler och sätta filrättigheter är inte tillräckligt för att förhindra två lokala användare att dela hemlig information mellan varandra. I detta dokument beskrivs några alternativ av gömda kanaler för att kringgå satta restriktioner samt eventuella motmedel för att förhindra användningen av dessa.

1. Inledning

I dagens samhälle är ofta information lagrad i digital form. Att bevara konfidentialiteten för hemlig information är betydligt svårare på digital form än med dess analoga motsvarighet, till exempel papper. Papper kan låsas in i kassaskåp dit endast betrodda har tillgång. Givetvis kan detta göras även med datorsystem, låsa in dem i bunkrar dit endast betrodda har tillträde. Oftast gäller dock att ett datorsystem hanterar flera olika användare samtidigt. I det fall då två användare med olika rättigheter på ett och samma system ska begränsas att inte dela information med varandra på grund av konfidentialitetsskäl uppstår ett problem. På grund av att användare i samma system delar på dess resurser, så som CPU, minne med mera, kommer det att vara näst intill omöjligt att förhindra delning av information mellan två användare. Om det är praktiskt omöjligt att helt förhindra delning av information på grund av delade resurser, skall alla eventuella kanaler som utnyttjar dessa resurser göras så pass långsamma att det är meningslöst för användarna att använda dessa för informationsdelning. Detta problem finns inom civila så väl som i militära system. Företagshemligheter för civila företag måste behållas hemliga och än viktigare är att försvarshemligheter som kan hamna i orätta händer hålls konfidentiella.

Syftet med detta dokument är att undersöka eventuella gömda kanaler som finns i ett MLS-system simulerat på operativsystemet FreeBSD [1], version 6.2. Simuleringen utförs genom att skapa två användare med rättigheter som efterliknar en MLS-miljö av modellen Bell-LaPadula. Funna gömda kanalers kapacitet och hur svårt det är att upptäcka dem kommer att mätas eller diskuteras. Eventuella motmedel för hur skapandet av dessa kanaler kan förhindras kommer även diskuteras samt vilken inverkan ett sådant motmedel skulle ha på systemet.

2. Bakgrund

Detta avsnitt beskriver de grundläggande koncept som hanteras genom hela detta dokument.

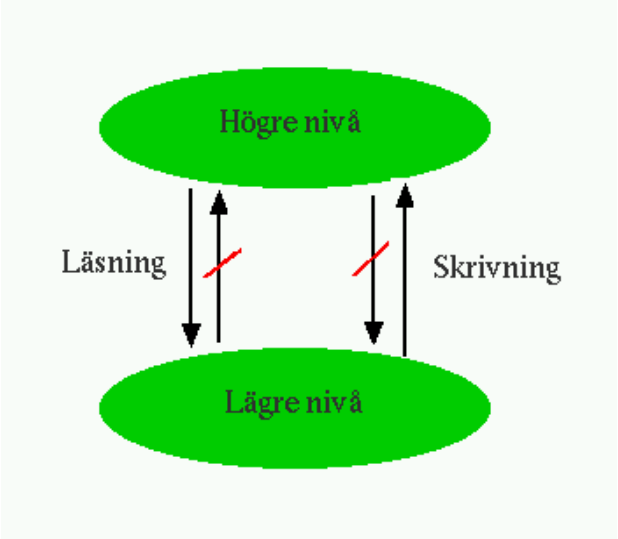
2.1 MLS

MLS är en förkortning för Multilevel Security och är ett regelverk för hur ett system ska hantera rättigheter för läsning samt skrivning av objekt i ett fleranvändar-system. Ett objekt kan vara en fil, en resurs som till exempel en skrivare eller en socket med mera. Detta görs genom att införa säkerhetsnivåer för både objekt och användare, där varje objekt och användare måste tillhöra en och endast en av säkerhetsnivåerna. Användare i ett MLS-system har ingen möjlighet att själva ändra några rättigheter för objekt, detta är den stora skillnaden mellan rättigheter i ett traditionellt fleranvändarsystem där DAC (Descretionary Access Control) används och ett MLS-system. En användare kan inte heller själv ändra vilken säkerhetsnivå denne tillhör, vilket också är fallet med traditionella fleranvändarsystem. Hur åtkomst för objekt ska hanteras bestäms av en så kallad modell, en sådan modell kan bevara antingen konfidentialitet, integritet eller de båda. Exempel på modeller är Bell-LaPadula, Biba, Clark-Wilson och Chinese Wall. I detta dokument kommer endast modellen Bell-LaPadula att hanteras.

2.2 Bell-LaPadula

Bell-LaPadula är en MLS-modell utvecklad av David Elliot Bell och Len LaPadula under 1970-talet då de hade som uppdrag att ta fram en MLS-modell åt U.S Air force [2]. Modellen är framtagen för att säkerställa att konfidentialitet för objekt i ett MLS-system bevaras. I praktiken innebär detta att en användare av lägre säkerhetsnivå ej kan läsa objekt av en högre säkerhetsnivå och att en användare av högre säkerhetsnivå ej kan skriva objekt till en lägre säkerhetsnivå än sin egen. Fördelen med detta är att en användare av en högre säkerhetsnivå inte kan dela med sig information till en annan användare med lägre säkerhetsnivå genom att ändra rättigheter för objekt. Dessutom skyddar detta mot trojaner och virus som annars skulle kunna ändra rättigheter för objekt så att en användare

av lägre säkerhetsnivå skulle kunna få tillgång till detta objekt. Figur 1 nedan visualiserar modellen Bell-LaPadula.



Figur 1. Policys i modellen Bell-LaPadula.

2.3 Gömda kanaler

En gömd kanal (eng. covert channel) är en kommunikationsväg som har som syfte att kunna överföra information utan att den upptäcks av någon utomstående. Då kanalen syftar till att vara gömd finns starka begränsningar vad gäller bandbredd, eftersom det är större risk att kanalen upptäcks om den använder mycket resurser. Detta kommer även naturligt i de flesta implementationer eftersom möjligheten att gömma information bland befintlig data ofta är väldigt begränsad. Gömda kanaler kan vara lokala i ett fleranvändar-system eller användas över nätverk. Gömda kanaler i nätverk innebär oftast att information göms i befintlig trafik till exempel genom att manipulera headers i nätverksprotokoll eller genom att skicka förutbestämt mönster av data.

2.3.1 Gömda kanaler i MLS-system

Gömda kanaler i ett MLS-system innebär i de flesta fall att den satta säkerhetspolicyn kringgås. Detta kan oftast göras genom att utnyttja det faktum att delade resurser alltid finns i ett fleranvändar-system. Delade resurser kan vara både hårdvaruresurser så som minnen och hårddiskar och operativsystemsspecifika resurser som till exempel filsystem, växlingsfiler, schemaläggning och så vidare.

3. Undersökta kanaler

Här beskrivs de gömda kanaler som undersökts samt resultat av undersökningen.

3.1 Åtkomsttider för delade resurser

I de flesta filsystem lagras information om senaste åtkomsttid för samtliga filer och kataloger. Detta kan utnyttjas som en gömd kanal för att överföra information mellan två parter [3]. Detta kan genomföras på följande sätt: Två parter väljer en fil eller katalog att använda, sedan synkroniserar de med varandra på något sätt för att bestämma när överföringen skall börja. Mottagaren registrerar nuvarande åtkomsttid för objektet och väntar sedan ett förutbestämt intervall innan en ny åtkomsttid registreras. Sändaren läser sedan det valda objektet så att dess åtkomsttid ändras. Detta skulle kunna innebära att skicka en etta, skulle sändaren vilja skicka en nolla läses inte objektet varvid åtkomsttiden ej ändras. Mottagaren kan nu, genom att jämföra om åtkomsttiden ändrats sedan föregående kontroll, avgöra om en etta eller nolla sänts.

3.1.1 Kapacitet

Upplösningen för tidsstämpeln för senaste åtkomsttiden för en fil i FreeBSD är i sekunder. En kanal kommer således att kunna överföra 1 bps (bit per sekund) per fil eller katalog som används som kanal. Om åtta filer eller kataloger används som transmissionsmedium kommer kanalen således att kunna överföra 1 Bps (byte per sekund), vilket innebär att två parter kan överföra ett tecken i sekunden. Om de kommunicerande parterna bestämmer sig för att koda om all text till gemener (och om man bortser från siffror och andra specialtecken) behövs endast 5 bitar för att kunna sända samtliga bokstäver i alfabetet. Görs detta fås en nästan dubbelt så snabb kanal jämfört med en kodning med 8 bitar per tecken. Ett problem som kan uppstå när flera filer eller kataloger används som transmissionsmedium (framför allt om kanalen ska användas på ett system med många användare) är den ökade risken till att en annan användare i systemet läser ett eller flera av de valda objekten. Åtkomsttiden för objektet kommer då att ändras som vanligt och (troligtvis) generera ett fel i överföringen för de två kommunicerande parterna. Problemet kan delvis lösas genom att skicka med paritetsbitar, dock medför extra paritetsbitarna att överföringshastigheten minskar. För att minska sannolikheten att en annan användare ska störa en överföring, bör objekt med låg åtkomstfrekvens väljas som transmissionsmedium. Väljs objekt med låg åtkomstfrekvens fås dock en annan ej önskvärd effekt, vilken beskrivs i avsnittet 3.1.3 Motmedel.

3.1.2 Begränsningar

En förutsättning för att denna kanal ska fungera är att de två kommunicerande parterna delar åtminstone en fil eller katalog som de båda har läsrättigheter till och att detta objekt inte används allt för ofta av övriga användare i systemet. Oftast finns det gott om systemfiler som alla användare måste ha läsrättigheter till, alltså bör detta sällan vara någon större begränsning.

Eftersom upplösningen på tidsstämplar för filer eller kataloger är sekunder är den maximala

överföringshastigheten 1 bps för varje objekt som används som kanal, denna begränsning går inte att komma ifrån. Eventuellt kan det finnas filsystem som har högre upplösning än sekunder, men det är inget som författarna av detta dokument känner till.

På ett system med många användare kan denna kanal bli märkbart begränsad på grund av att antalet passande objekt (objekt som inte läses ofta) blir betydligt färre än i ett system med få användare.

3.1.3 Motmedel

Det mest självklara motmedlet är att helt enkelt stänga av åtkomsttidsloggningen i systemet. Då finns ingen som helst möjlighet att använda denna metod för att överföra data. Skulle administratören av ett system vara intresserad av åtkomsttider för filer och kataloger av någon anledning är detta en dålig lösning. Alternativet till att stänga av åtkomsttidsloggningen helt skulle kunna vara att logga onaturligt höga åtkomstfrekvenser per användare. Genom att logga åtkomstfrekvenser skulle en administratör även kunna avslöja vilka de eventuella "spionerna" är. Har de kommunicerande parterna valt objekt som sällan används i ett system (vilka oftast är önskvärt för att få så låg felfrekvens som möjligt) för att kommunicera blir det ännu lättare för en administratör att upptäcka detta.

Om det inte finns några filer som de båda användarna har tillgång till kan de inte kommunicera med varandra. För att förhindra att användare av ett system delar på några som helst filer eller kataloger kan varje användare "låsas in" på en egen virtuell maskin. Detta kan i FreeBSD göras med kommandot 'jail' [4]. När användarna "låsts in" har de inte åtkomst till några gemensamma filer och kan således inte kommunicera med varandra.

3.1.4 Implementering

En implementering av ovan beskriven kanal finns bifogad i Appendix A. Koden är kompilerad och testad under FreeBSD 6.2. Implementeringen består av ett sändarprogram och ett mottagarprogram, där båda tar en starttid som argument vid uppstart av programmet. Detta görs eftersom de båda programmen måste synkronisera med varandra. De två programmen använder sig sedan av systemklockan för att synkronisera med varandra. Sändarprogrammet tar även en fil som argument, innehållet i filen är den data som kommer att skickas över kanalen. Innehållet antas vara en text skriven med ascii-tecken (programmet är endast testad med ascii som indata). Programmen använder katalogerna "/boot", "/etc", "/usr/lib" och "/lib" som kommunikationskanaler (vilket inte är några optimala kataloger att använda i ett system med många användare, ty dessa används ganska ofta). Sändarprogrammet omvandlar ett tecken från filen till bitmönster och öppnar sedan de kataloger som ges av bitmönstret. Om en bit är en etta skall en katalog öppnas, annars öppnas inte katalogen. Detta görs med alla tecken som finns i filen. Exempel: bitmönstret är "0101", de kataloger som kommer att öppnas blir därför "/etc" och "/lib". Skulle istället bitmönstret vara "1000", kommer endast "/boot" att öppnas. Mottagarprogrammet läser av katalogerna och ser om åtkomsttiden för någon katalog ändrats sedan föregående cykel, om så är fallet tolkas det som en etta, annars en nolla. Efter att ha mottagit åtta bitar omvandlar mottagarprogrammet de åtta bitarna till motsvarande ascii-tecken och skriver ut det till stdout. Sändningen avslutas då mottagarprogrammet erhåller åtta nollor i följd.

3.2 Skapande av existerande filer

Enligt Bell-LaPadula är det ej tillåtet för en användare att läsa från en högre nivå eller att skriva till en lägre nivå än vad användaren har. Det är dock möjligt för en användare att skriva till en högre nivå än vad denne själv har. Om två parter har skrivrättigheter till ett och samma ställe, till exempel en temp-katalog, kan de kommunicera genom att försöka skapa filer med samma namn. Detta görs genom att användaren med högre nivå skapar en fil vars namn användaren med lägre nivå vet om. Användaren med lägre nivå försöker sedan skapa en fil med samma namn och om filen redan existerar fås ett felmeddelande, vilket motsvarar en förutbestämd bit. Den användaren som lyckas skapa filen måste sedan ta bort den innan nästa bit kan överföras.

3.2.1 Kapacitet

Kapaciteten hos denna kanal beror på två huvudsakliga delar. Den första är hur bra upplösning hos systemklockan en användare har i userland, och avgör då hur storleksordningen för väntetiden mellan de två programmen kan väljas. I FreeBSD är denna upplösning i storleken mikrosekunder. Den andra delen är den tid det tar att skapa filer, vilket kan vara svårt att mäta då detta beror på hårdvara, vilket filsystem som används, systembelastning, eventuella inställningar som begränsar hur många filer en användare kan skapa, med mera. Dessa två delar är även beroende av varandra; om man väljer att skapa väldigt många filer kan detta ta ett antal mikrosekunder, vilket innebär att man måste öka den tid som de två programmen väntar så att de inte kommer ur synk. Det bör nämnas att avsevärt högre överföringshastighet går att få än i implementationen som presenteras här, vilken har en överföringshastighet på 1/3 byte per sekund.

3.2.2 Begränsningar

Denna gömda kanal förutsätter att användare av två olika nivåer har skrivrättigheter till en gemensam katalog, till exempel en temp-katalog. Denna metod är realiserbar i ett MLS-system som använder modellen

Bell-LaPaula och kan eventuellt även fungera i andra modeller beroende på de policys som dessa definerar.

3.2.3 Motmedel

Då denna gömda kanal innebär skapande och borttagande av flertalet filer i en snabb takt är den ganska lätt att finna för en administratör som aktivt letar efter suspekta aktiviteter i ett system. En administratör skulle kunna skriva ett simpelt bakgrundsprogram som kontrollerar delade kataloger efter skapande av många filer av en och samma användare och låta programmet vidta åtgärder i realtid.

3.2.4 Implementering

Implementationen som återfinnes i Appendix B använder sig av åtta olika filer för överföringen. Att just åtta filer valdes beror på att varje enskild fil då kan representera en bit i varje tecken som överförs, vilket underlättade vid implementeringen. Högnivåanvändaren samt lågnivåanvändaren anger en överenskommen tid då överföringen skall starta som ett argument till programmet. Överföringen sker sedan för varje byte enligt figur 2.

Tidpunkt	**Sändare**	**Mottagare**
T1	Skapa filer för de bitar som är ettor.	Vänta.
T1+1s	Vänta.	Försök skapa alla filer och notera vilka som existerar.
T1+2s	Ta bort eventuella skapade filer.	Ta bort eventuella skapade filer.

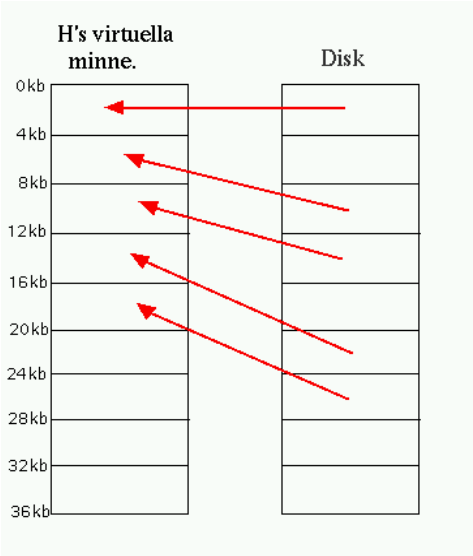
Figur 2. Beskrivning av implementation.

Det tar alltså 3 sekunder att överföra ett tecken. Anledningen till att implementationen är gjord på sekundnivå är att det är avsevärt lättare då man inte behöver addera sekunder och mikrosekunder för att räkna ut nästa tidpunkt i överföringen.

3.3 Minnesmappning

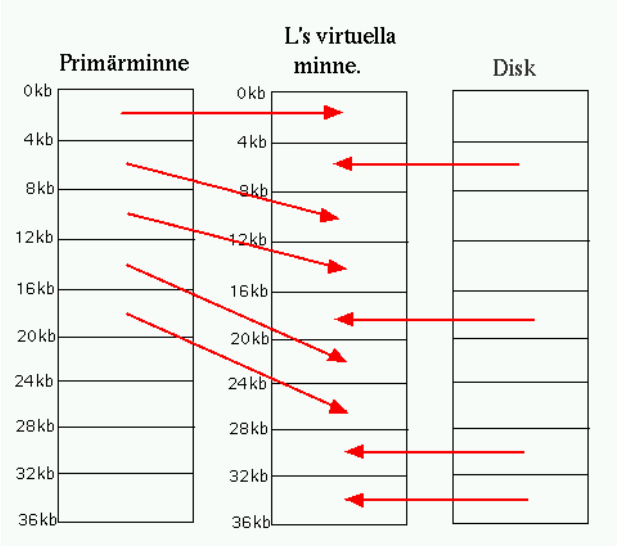
En resurs som princip alltid är delad i ett fleranvändar-system är primärminnet. När en användare öppnar en fil så lagrar operativsystemets kärna åtminstone så mycket av filen som motsvarar en sida av sidtabellen (eng. page table) i primärminnet. Antag att en sida i det virtuella minnets sidtabell är 4 kb. Vid läsning av första byten i en fil läses 4 kb in med automatik till primärminnet vilket refererande sida i sidtabellen sedan mappas mot [5]. Kärnan lagrar även den öppna filens inode-nummer i tabellen över öppna filer. Om ytterliggare en användare öppnar samma fil kommer kärnan upptäcka att filen redan är öppnad genom att jämföra filens inode-nummer med de inode-nummer som finns i tabellen över öppna filer. Genom detta förfarande kan kärnan låta den nya läsningen göras direkt från primärminnet istället för att läsa in samma fil från disk igen. Om den första användaren endast läst i de första 4 kb av filen och den andra användaren läser utöver de 4 kb som finns lagrade i primärminnet, kommer ett sidfel (eng. page fault) att genereras och kärnan kommer då läsa in ytterliggare 4 kb av filen till primärminnet.

Eftersom en läsning från disk tar betydligt längre än att läsa direkt från primärminnet skulle en gömd kanal kunna skapas enligt följande [6]:
Användare H och användare L har båda tillgång till en referensfil, den bör vara stor eftersom varje sida i sidtabellen kommer att motsvara en bit. Användare H tillhör en högre säkerhetsnivå än användare L. Användare H läser in 4 kb av filen, hoppar över 4 kb, läser ytterliggare 8 kb, hoppar över 4 kb och läser till sist in 8 kb, se figur 3 nedan. (Egentligen räcker det med att läsa första byten i varje 4 kb's segment i filen eftersom att det virtuella minnet endast hanterar hela sidor).



Figur 3. Användare H's läsning av fil.

Användare L läser nu hela filens innehåll. De sidor av filen som användare H har läst ligger redan i primärminnet och L kommer således att läsa dessa sidor direkt ur primärminnet istället för från disk. Se figur 4 nedan.



Figur 4. Användare L's läsning av hela filen.

Genom att mäta den tid varje sida tar att läsa in från filen, kan användare L se vilka sidor av filen som H har läst in. De sidor som lästs kan till exempel motsvara ettor och de ej lästa nollor - användarna H och L kan nu kommunicera med varandra.

3.3.1 Utredning

Denna kanal fungerar utmärkt i teorin men är betydligt svårare att implementera i praktiken. Skulle en kanal skapas enligt denna metod skulle överföringshastigheter i storleksordningen flera hundra bitar per sekund kunna fås [6]. Problemet med denna kanal ligger bland annat i att kunna mäta tider med tillräckligt hög upplösning i userland samt att mottagarprogrammet påverkas av en eventuell context switch. Antag att mottagarprogrammet startar en tidtagning innan den börjar läsa en sida. Om mottagarprogrammet blir avbruttet av en context switch precis efter att tidtagningen startat kommer tiden för sidläsningen att bli väldigt lång i förhållande till om mottagarprogammet inte blivit avbrutet. Mottagarprogrammet skulle i detta fall antagligen tolka det som en diskläsning, vilket det i själva verket kanske inte alls är, varpå en felaktig bit fås. För att få ett bra fungerande mottagarprogram skulle avbrott behöva stängas av under den tiden som mottagarprogammet gör sin tidtagning och läsning, för att på så sätt slippa bli avbrutet av en context switch. Dock så skulle detta kräva superuser-rättigheter, men skulle en användare ha dessa rättigheter så kan denne kringgå MLS-systemet ändå.

4. Slutsats

Att eliminera alla möjligheter till gömda kanaler vid design av MLS-system är en otroligt svår, om inte omöjlig uppgift. Så fort ett system skall kunna hantera flera användare uppstår ett behov av delade resurser, vilket är förutsättningen för att en gömd kanal skall kunna skapas. Ett system som är designat att eliminera så många kända gömda kanaler som möjligt får även lidande prestanda eftersom att man då undviker att dela resurser mellan användare i den mån som detta är möjligt. En anledning till att system delar resurser mellan användare är just av prestandaskäl; det är ju till exempel slöseri med minne att ha samma saker på flera olika ställen i minnet. Som visats i denna rapport behöver en simpel gömd kanal inte vara svår att implementera, det är tvärtom en ganska lätt uppgift. Dock är ofta dessa simpla gömda kanaler relativt lätta att finna för en administratör som aktivt letar efter gömda kanaler, vilket kan, som beskrivits tidigare göras genom att skriva en process som körs i bakgrunden och regelbundet letar efter kända mönster. Gömda kanaler är ett stort område, vilket alltid är aktuellt vid utveckling av säkra system och är något som är viktigt att ha i åtanke.

Referenser

[1] FreeBSD, “The FreeBSD foundation”, http://www.freebsd.org, 2007.

[2] Bell, D.D. and L.J. La Padula (1974). Secure Computer System: Unified Exposition and Multics Interpretation, ESD-TR-75-306. Bedford, MA: ESD/AFSC, Hanscom AFB. http://csrc.nist.gov/publications/history/bell76.pdf , 2007.

[3] V. Petricek (2001), “Information hiding and covert channels”, Proceedings of 14th Annual Student Conference Week of Doctoral Students (WDS2001),Prague, http://kocour.ms.mff.cuni.cz/~petricek/papers/covertalk/ih-wds.pdf, 2007.

[4] FreeBSD 6.2, jail utility, http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html, 2007.

[5] A. Silberschatz, P.B. Galvin, G. Gagne (2004), “Operating system concepts, seventh edition”, John Wiley & Sons, Inc. ISBN 0-471-69466-5.

[6] C. Percival (2005), “Cache missing for fun and profit”, IRMACS Center, Simon Fraser University, Burnaby, BC, Canada, http://www.daemonology.net/papers/htt.pdf, 2007.

Appendix A

---- *Begin atime_write.c* ----

```
/* atime_write.c
 *
 * To compile: gcc -o send atime_write.c -lm
 *
 * Written by,
 * jerer606@student.liu.se
 * jonte466@student.liu.se
 */

#include <stdio.h>
#include <fcntl.h>
#include <sys/time.h>

/* Delay before starting to send */
#define STARTUP_DELAY   1

/* Delay between sending bits */
#define WRITE_DELAY     2

/* Number of channels */
#define NUMCHANNELS     4

/* Macros for common things */
#define GETBIT(a, b) (a&(int)pow(2,b)) > 0 ? 1 : 0

#define SEND(a) if (a == 0 || a == 4) read(fd[0], &c, 1); else if \
                (a == 1 || a == 5) read(fd[1], &c, 1); else if \
                (a == 2 || a == 6) read(fd[2], &c, 1); else read(fd[3], &c, 1)

#define WAIT()  starttime += WRITE_DELAY; do { time(&currtime); usleep(10); } \
                while (starttime > currtime)

char    *channels[NUMCHANNELS] = { "/boot", "/etc", "/usr/lib", "/lib" };

int
main(int argc, char *argv[])
{
        int     i, j;
        size_t  len;
        int     fd[4];
        u_char  c;
        u_char  sendbuf[512];
        FILE    *fp;
        struct  tm tm;
        time_t  currtime, starttime;

        if (argc < 3) {
                printf("usage: %s <input file> HH:MM:SS\n", argv[0]);
                exit(1);
        }

        /* Try to open the file containing the data to be sent */
        if ((fp = fopen(argv[1], "r")) == NULL) {
                perror("open()");
                exit(1);
        }

        /* Read the file containing the message to send */
        len = fread(sendbuf, sizeof(u_char), sizeof(sendbuf), fp);
        sendbuf[len-1] = '\0';

        printf("[*] Start time: %s\n", argv[2]);

        /* Get the current time, we do this so we don't have to set all the
           fields in tm before we call strptime() below. */
        time(&starttime);
        tm = *localtime(&starttime);

        /* We want to use the time specified in the command line argument */
        if (strptime(argv[2], "%H:%M:%S", &tm) == NULL) {
                perror("strptime()");
                exit(1);
```

```
        }

        /* Convert to epoch time */
        starttime = mktime(&tm);

        /* We have a small delay compared to the receiver */
        starttime += STARTUP_DELAY;

        /* Wait until the specified time before we start to send */
        do {
                time(&currtime);
                usleep(10);
        } while (starttime > currtime);

        printf("[*] Sending message: ");
        fflush(stdout);

        /* Open the target directories */
        for (i = 0; i < NUMCHANNELS; i++) {
                if ((fd[i] = open(channels[i], O_RDONLY|O_NONBLOCK)) == -1) {
                        perror("open()");
                        exit(1);
                }
        }

        /* Send the data */
        for (i = 0; i < len; i++) {
                for (j = 0; j < 4; j++) {
                        if (GETBIT(sendbuf[i], j))
                                SEND(j);
                }

                WAIT();

                for (j = 4; j < 8; j++) {
                        if (GETBIT(sendbuf[i], j))
                                SEND(j);
                }

                printf("%c", sendbuf[i]);
                fflush(stdout);

                WAIT();
        }

        printf("\n[*] Data has been sent!\n");

        /* Clean up */
        for (i = 0; i < NUMCHANNELS; i++) {
                close(fd[i]);
        }

        fclose(fp);

        return 0;
}
```

---- *End atime_write.c* ----

---- *Begin atime_read.c* ----

```
* atime_read.c
 *
 * To compile: gcc -o receive atime_read.c
 *
 * Written by,
 * jerer606@student.liu.se
 * jonte466@student.liu.se
 */
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>

/* specifies the time to wait for the sender to send its bits
   in each cycle. In seconds. */
const int sleeptime = 2;
```

```
/* to keep info about out channel objects */
struct ch {
  time_t atime_old, atime_new;
  struct stat sb;
};

/* specifies what catalogs or files we will be using to communicate */
const char *channels[] = { "/boot", "/etc", "/usr/lib", "/lib" };

/*
 *  Converts an 8 bytes long string that must contain
 *  only 1:s and 0:s, to the corresponding integer value
 *  using the last byte in the string as the MSB.
 */
char convert(char *letter) {

        char b;

        bzero(&b,0);

        b += letter[7]*128;
        b += letter[6]*64;
        b += letter[5]*32;
        b += letter[4]*16;
        b += letter[3]*8;
        b += letter[2]*4;
        b += letter[1]*2;
        b += letter[0]*1;

        return b;
}

int
main(int argc, char *argv[])
{
        int i, j;
        struct ch channel[4];
        time_t currtime, starttime;
        struct tm tm;
        unsigned char received_char[8], tkn;
        if(argc < 2 ) {
          printf("usage: %s hh:mm:ss\n", argv[0]);
          exit(1);
        }

        // get start time
        time(&starttime);
        tm = *localtime(&starttime);

        strptime(argv[1], "%H:%M:%S", &tm);

        starttime = mktime(&tm);

        printf("Waiting for start time to arrive...\n");

        /* do not start until specified time has arrived */
        do {
          time(&currtime);
          usleep(10);
        } while (starttime > currtime);

        printf("Starting to receive!\n");
        printf("Receiving message: ");
        fflush(stdout);

        do {

          for(j = 0; j < 2 ; j++) {

            // sets the next time we will check atime for our channels
            starttime += sleeptime;

            // read status for our channels
            for(i = 0; i < 4 ; i++) {
              if ((stat(channels[i], &channel[i].sb)) == -1) {
```

```
            perror("stat() channel[i]");
            exit(1);
          }
          // save atime
          channel[i].atime_old = channel[i].sb.st_atime;
        }

        // wait until read time arrives so that the
        // sender gets time to change atime of the channels
        do {
          time(&currtime);
          usleep(10);
        } while (starttime > currtime);

        // read status again
        for(i = 0; i < 4 ; i++) {
          if ((stat(channels[i], &channel[i].sb)) == -1) {
            perror("stat() channel[i]");
            exit(1);
          }
          // save new atime
          channel[i].atime_new = channel[i].sb.st_atime;

          // write the received bits
          if(channel[i].atime_new > channel[i].atime_old) {
            // received a 1
            #ifdef DEBUG
              printf("received: 1\n");
            #endif
            received_char[i+(j*4)] = 1;
          }
          else {
            // received a 0
            #ifdef DEBUG
              printf("received: 0\n");
            #endif
            received_char[i+(j*4)] = 0;
          }

        }

        // if we have received 8 bits
        if(j == 1) {
          tkn = convert(received_char);

          // if message is finished
          if(tkn == 0) {
            printf("\nMessage received succesfully! Exiting...\n");
            exit(0);
          }
          printf("%c", tkn);
          fflush(stdout);
        }
      }

    } while(1);


    return 0;
}
```

---- *End atime_read.c* ----

Appendix B

---- *Begin cc_file.c* ----

```
/* Covert channel using file creation.
 *
 * To build the sender: gcc cc_file.c -DSENDER -lm -o sender
 * To build the receiver: gcc cc_file.c -DRECEIVER -lm -o receiver
 *
 * Written by,
 * jerer606@student.liu.se
 * jonte466@student.liu.se
 */
```

```
#include <stdio.h>
#include <sys/types.h>
#include <time.h>
#include <fcntl.h>
#include <math.h>

/* We use 8 channels.. one for each bit of a character */
#define NUMCHANNELS     8

/* Each character takes 3*DELAY seconds to transmit */
#define DELAY   1

/* Macro to get a specified bit */
#define GETBIT(a, b)    (a&(int)pow(2,b)) > 0 ? 1 : 0

/* Macro for waiting */
#define WAIT(a)         starttime += a; do { time(&currtime); usleep(10); } \
                        while (starttime > currtime);

/* The channels we use */
char *channels[NUMCHANNELS] = \
        { "/tmp/test111", "/tmp/test222", "/tmp/test333", "/tmp/test444", \
         "/tmp/test555", "/tmp/test666", "/tmp/test777", "/tmp/test888"};

/* We must keep track of which files we successfully created
   so that we can delete them later. */
int created[NUMCHANNELS];

void
clean()
{
        int i, err;

        /* Remove all created files */
        for (i = 0; i < NUMCHANNELS; i++) {
                if (created[i] == 1) {
                        if ((err = remove(channels[i])) == -1) {
                                perror("remove()");
                                exit(1);
                        }
                }
                /* Reset the list */
                created[i] = 0;
        }
}

int
create(int target)
{
        int fd;

        /* Try to create the file */
        if ((fd = open(channels[target], O_CREAT|O_EXCL)) == -1) {
#ifdef  SENDER
                /* Shouldn't happen, bail out */
                perror("open()");
                exit(1);
#elif   RECEIVER
                /* The file already exists - return that this bit
                   should be a 1 */
                return 1;
#endif
        } else {
                /* We successfully created the file, keep
                   track of it in our list. */
                created[target] = 1;
        }

        close(fd);

        return 0;
}

int
main(int argc, char *argv[])
```

```
{
	int		i, j;
	struct tm	tm;
	time_t		currtime, starttime;

#ifdef  SENDER
	size_t		len;
	u_char		sendbuf[512];
	FILE		*fp;
#elif   RECEIVER
	u_char		b;
#endif

	/* We dont have any created files yet */
	for (i = 0; i < NUMCHANNELS; i++)
		created[i] = 0;

#ifdef  SENDER
	if (argc < 3) {
		printf("usage: %s HH:MM:SS <file to send>\n", argv[0]);
		exit(1);
	}
#elif   RECEIVER
	if (argc < 2) {
		printf("usage: %s HH:MM:SS\n", argv[0]);
		exit(1);
	}
#endif

	printf("[*] Start time: %s\n", argv[1]);

	time(&starttime);
	tm = *localtime(&starttime);

	if (strptime(argv[1], "%H:%M:%S", &tm) == NULL) {
		perror("strptime()");
		exit(1);
	}

	/* Convert to epoch time */
	starttime = mktime(&tm);

#ifdef  SENDER
	/* Try to open the file to send */
	if ((fp = fopen(argv[2], "r")) == NULL) {
		perror("open()");
		exit(1);
	}

	/* Read in the contents of the file */
	len = fread(sendbuf, sizeof(u_char), sizeof(sendbuf), fp);
	sendbuf[len -1] = '\0';

	fclose(fp);
#elif   RECEIVER
	/* We have a small delay compared to the sender */
	starttime += DELAY;
#endif

	/* Wait until the specified time before we start */
	do {
		time(&currtime);
		usleep(10);
	} while (starttime > currtime);

#ifdef  SENDER
	printf("[*] Sending message:\n");
	fflush(stdout);

	for (i = 0; i < len; i++) {
		/* Get the individual bits of each character */
		for (j = 0; j < 8; j++) {
			if (GETBIT(sendbuf[i], j))
				create(j);
		}
```

```
            printf("%c", sendbuf[i]);
            fflush(stdout);

            WAIT(2*DELAY);
            clean();
            WAIT(DELAY);
        }

        printf("\n[*] Message was sent!\n");
#elif   RECEIVER
        printf("[*] Receiving message:\n");
        fflush(stdout);

        while (1) {
            b = 0;

            /* Build the character from the received bits */
            for (i = 0; i < 8; i++) {
                b += pow(2, i)*create(i);
            }

            WAIT(DELAY);
            clean();
            WAIT(2*DELAY);

            /* We received NULL, transmission is complete */
            if (b == 0)
                break;

            printf("%c", b);
            fflush(stdout);
        }

        printf("\n[*] Message recieved!\n");
#endif

        /* Clean up */
        clean();

        return 0;
}
```
---- *End cc_file.c* ----