Deluppgift 15 Filhantering: create, open, read, write, close, remove

Saker som kan vara bra att känna till

Från föregående uppgift vet du att tangentbordet identifieras av fil-deskriptor (`fd`) STDIN_FILENO och att skärmen identifieras av STDOUT_FILENO. Bägge deklararade i `lib/stdio.h`.

Funktioner för att manipulera katalogträdet (listan på filnamn lagrad på disk) återfinns i `filesys/filesys.h`. Funktioner finns för att *lokalisera* (söka, öppna), *skapa* (lägga till), och *ta bort* filer i katalogträdet. PintOS implementation av filsystem är begränsad till max 16 filer, inga kataloger, och korta filnamn utan konstiga tecken. Filstorleken går inte att utöka, utan måste anges då filen skapas.

Funktioner för att hantera filer som redan är öppnade återfinns i `filesys/file.h`. Funktioner finns för att *läsa, skriva, ta reda på läs/skrivposition, ange läs/skriv position, ta reda på filens storlek*, samt *stänga* filen och frigöra de resurser PintOS allokerat för filen i fråga.

Borttagning av filer hanteras speciellt i den befintliga implementationen. När filen tas bort försvinner den direkt ur katalogen, så den inte går att öppna mer, men avallokering av diskblock sker inte direkt, utan först när alla som använder samma fil stängt sin fil. Den sista som stänger kommer att markera filens block som lediga igen. Detta gör att borttagning av filer som är öppna och används av andra processer sker korrekt.

Alla filer som öppnats av en viss process skall *alltid* stängas då *den* och *endast den* processen avslutar. Detta för att undvika minnesläckage i operativsystemet. En process kan avsluta genom att returnera från main, via systemanropet exit, eller genom att PintOS dödar processen till följd av något programmeringsfel (i processen). I alla tre fall skall kvarvarande öppna filer stängas.

Beskrivning av uppgiftens systemanrop

Systemanropen för filhantering är deklarerade i `lib/user/syscall.h` och används av användarprogrammen därefter:

`int open (const char *file);`

Skall söka i katalogträdet efter en fil med namnet angivet i `file`. Om filen finns hämtas referensen till filen (`struct inode*`) och lagras i systemets inodelista. Om filen redan öppnats av någon process hämtas referens till filen direkt från inodelistan. Referensen till filen (`struct inode*`) lagras tillsammans med information om läs och skrivposition i `struct file*` för snabb och effektiv åtkomst vid senare läsningar eller skrivningar till filen. Notera att det kan komma skapas många öppna filer av typ `struct file*` för varje filreferens av typ `struct inode*`. *Ovan löses i befintliga funktioner, studera koden*.

För att dölja all kernel-data från användar-programmet knyts informationen (`struct file*`) till ett heltal som benämns fildeskriptor (`fd`). Det är detta heltal som returneras till användarprogrammet. Minus ett (-1) returneras om angiven fil inte finns. Några fd är reserverade för tangentbord och skärm. Du måste hålla reda på vilken fil som är knuten till vilken fildeskriptor för varje filöppning, samt vilken process som gjorde filöppningen. Alla `fd` en process fått via anrop av `open` måste till slut stängas oavsett om processen anropar `close` eller avslutar utan anrop av `close`, oavsett hur processen avslutas.

`int read (int fd, void *buffer, unsigned length);`
: Läs `length` tecken från resursen `fd` och lagra i `buffer`. Returnera antal lästa tecken. `fd` kan vara `STDIN_FILENO`(tangentbord) eller en `fd` som processen tidigare fått i retur från `open`, men ännu inte skickat som argument till `close`. `STDOUT_FILENO` är ett ogiltigt `fd` vid anrop till `read`.

`int write (int fd, const void *buffer, unsigned length);`
: Skriv `length` tecken lästa från `buffer` till resursen `fd`. Returnera antal skrivna tecken. `fd` kan vara `STDOUT_FILENO` (skärm) eller en `fd` som processen tidigare fått i retur från `open`, men ännu inte skickat som argument till `close`. `STDIN_FILENO` är ett ogiltigt `fd` vid anrop till `write`.

`void close (int fd);`
: Stäng (avallokera kernelresurser) den fil som identifieras av `fd`. Minne för `struct file*` och ibland `struct inode*` skall återlämnas. *Studera hur detta redan fungerar.* Ett giltigt `fd` måste ha erhållits som returvärde vid tidigare anrop av `open` och ännu inte skickats som argument till `close`. Alla `fd` en process fått via anrop av `open` måste till slut stängas oavsett om processen anropar `close` eller avslutar utan anrop av `close`, oavsett hur processen avslutar. En process får bara lov att stänga filer som den själv har öppnat.

`bool remove (const char *file);`
: Sök i katalogträdet efter en fil med namnet angivet i `file`. Om sådan finns raderas namnet och referensen till filen i katalogträdet, men inga diskblock avallokeras förrän alla som har filen öppen har stängt filen. *Ovan löses i befintliga funktioner.* Returnerar `true` om filnamnet raderades från katalogträdet.

`bool create (const char *file, unsigned initial_size);`
: Skall reservera diskblock för en fil med storlek `initial_size` och lägga till en referens till filen i katalogträdet under namnet angivet i `file`. Returnera `true` om operationen lyckas.

Uppgift

Implementera och testa systemanropen ovan. Tänk på att en viss process endast skall kunna läsa från, skriva till och stänga filer som den har öppnat, och tänk på att alla processens filer måste stängas exakt en gång, senast då processen avslutas.

Planera din implementation noga. Skriv funktioner för kod du behöver upprepa mer än en gång, t.ex. kontroll att en `fd` är giltig och öppnad av processen. Du kan behöva modifiera föregående uppgifter och befintlig kod. Den associativa kontainer (`map`) som kopplar öppnade filer (`struct file*`) mot fildeskriptor (`int fd`) är lämplig att implementera i `userprog/flist.c`. Kopiera t.ex. över din kod från uppgift 6 (`map.c` och `map.h`) till `flist.c` och `flist.h` och anpassa för fillagring. Alternativt kan du lägga till `map.c` i `src/Makefile.build` och om du vet vad du gör anpassa för lagring av generisk pekare (`void*`).

Hanteraren för respektive systemanrop skrivs som förut i `userprog/syscall.c`. I uppgift 10 finns information om `process_cleanup` som du bör finna relevant, och i uppgift 12 (och 8) finns information om stacken (var hittar du parametrar? var placerar du returvärden för systemanrop?).

Filen `examples/file_syscall_tests.c` innehåller ett lämpligt testprogram för att testa systemanropen för filhantering. Observera att det även testar kommande uppgifter. Tänk på att `file_syscall_tests` är ett för långt filnamn för PintOS, se kommentar i filen för hur du kan starta användarprogrammet.