LINKÖPINGS UNIVERSITET

TDIU16: Process- och operativsystemprogrammering

Systemanrop för filhantering

Klas Arvidsson, Daniel Thorén, Filip Strömbäck

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2018

2020-03-06

1 Mål

I den här laborationen ska vi utöka systemanropen `read` och `write` så att de också fungerar för filer. För att det ska fungera behöver vi även några ytterligare systemanrop, nämligen `open`, `close`, `create` och `remove`. I slutändan vill vi att ett program ska kunna skapa en fil och skriva data till den, ungefär på det här sättet:

```
int main() {
  create("test", 12);            // Skapa en fil som är 12 bytes stor.
  int fd = open("test");         // Öppna filen.
  write(fd, "Hello World!", 12); // Skriv 12 bytes till filen.
  close(fd);                     // Vi är klara, stäng filen.
  return 0;
}
```

Vi vill såklart också att program ska kunna läsa från filer och ta bort filer med `read` och `remove` på liknande sätt.

2 Testprogram

Filen `examples/file_syscall_tests.c` innehåller ett lämpligt testprogram för att testa systemanropen för filhantering. Tänk på att `file_syscall_tests` är ett för långt filnamn för Pintos, se kommentar i filen för hur du kan starta användarprogrammet.

3 Uppgift

Implementera och testa följande systemanrop. Exakt vad de ska göra finns i Pintos-Wiki under :

- `int open(const char *file);`
- `int read(int fd, void *buffer, unsigned length);`

  (lägg till stöd för filer)
- `int write(int fd, const void *buffer, unsigned length);`

  (lägg till stöd för filer)
- `void close(int fd);`
- `bool remove(const char *file);`
- `bool create(const char *file, unsigned initial_size);`

  Notera: Filsystemet i Pintos skiljer sig från andra system här. Alla filer i Pintos har en fix storlek (dvs. de kan inte växa när man lägger till mer data i slutet av dem). Därför måste man ange storleken när man skapar filen.
- `void seek(int fd, unsigned position);`
- `unsigned tell(int fd);`
- `int filesize(int fd);`

Tänk på att en viss process endast skall kunna läsa från, skriva till, och stänga filer som den har öppnat, och att alla filer måste stängast exakt en gång, senast då processen avslutas (även om `close` inte anropas, eller om den anropas flera gånger).

Stora delar av filsystemet finns redan implementerat i Pintos, se Pintos-Wiki under ”Systemanrop” i avsnittet ”Filsystem”. Målet med laborationen är alltså att låta program i usermode använda denna existerande funktionalitet på ett säkert sätt snarare än att implementera ett helt filsystem.

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 uppgiften ”Associativ container” (`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 Pintos-Wiki under rubriken ”Interna funktioner” i avsnittet ”Tråd- och processhantering i Pintos” finns information om `process_cleanup` som kan vara intressant.

4 Att tänka på

Här kommer en lista av saker som är bra att fundera över när du implementerar systemanropen:

- Vad är en fildeskriptor? Vad kan ett program förvänta sig av en fildeskriptor, och vad behöver därmed operativsystemet se till?
- Hur ska operativsystemet hantera fallet då en process försöker öppna en fil, men inte lyckas lägga in den i listan över öppna filer? Det är okej att operativsystemet får slut på resurser, men hur ska operativsystemet meddela detta?
- Hanterar implementationen att ett program råkar stänga samma fildeskriptor två gånger?
- Hanterar implementationen att ett program råkar glömma att stänga en fil?
- Finns det något sätt ett dåligt skrivet (eller elakt) program kan få systemet att krascha eller läcka resurser?