LINKÖPINGS UNIVERSITET

TDIU16: Process- och operativsystemprogrammering

Processhantering: `wait`

Klas Arvidsson, Daniel Thorén, Filip Strömbäck

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2018

2020-03-06

1 Mål

Målet med den här uppgiften är att göra klart processhanteringen i Pintos som vi började på i förra laborationen. Nu vill vi att nedanstående exempel ska fungera till fullo.

```
int main() {
  pid_t child = exec("sumargv 1 2 3"); // Starta en barnprocess som summerar argumenten
  int result = wait(child);            // Vänta på att den blev klar och hämta resultatet
  printf("Sum: %d\n", result);         // Skriv ut det.
  return 0;
}
```

2 Uppgift

Implementera systemanropet:

`int wait(pid_t pid);`

Gör detta genom att utöka informationen i listan med aktiva processer med den information som behövs för att spara `status` i processinformationen när en process avslutar. När en process anropar `wait` skall statusinformationen *som sparats eller kommer att sparas* för angiven barnprocess returneras från `wait` när den finns tillgänglig. Har du implementerat `exec` och `wait` korrekt bör detta vara enkelt, eftersom processinformationen hela tiden finns tillgänglig för både processen som anropar `wait` (föräldern) och processen som sparar sin `status` (barnet). Tänk noga igenom var du placerar synkroniseringsvariabler som behövs för att låta föräldern vänta tills barnet är klart. Det får inte finnas någon risk att föräldern väcks av något annat barn än det den väntar på. Och det får inte finnas någon risk att variablerna är borttagna när de behövs. Tänk också på att det bara är föräldraprocesser som får vänta på sina barn. Det ska alltså inte gå att vänta på vilka processer som helst med hjälp av `wait`.

Tänk på att placera din kod i filen `userprog/process.c`. Koden i `userprog/syscall.c` skall endast innehålla ett anrop till relevant funktion. Att du måste göra så beror på att den första processen inte startas via ett systemanrop.

2.1 Testning

Filen `examples/wait_test.c` innehåller ett enkelt testprogram för att testa så att systemanropen `exec` och `wait` verkar fungera som de ska på liten skala. Filen `examples/longrun.c` innehåller ett testprogram som stresstestar implementationen lite mer. Se kommentarer i testprogrammen för att se hur de kan startas.

Hittills har du vid varje Pintos-körning fått en felutskrift om att huvudprogrammet (main) försöker stänga av datorn innan alla trådar är klara. När din lösning fungerar korrekt skall du inte längre få *några ERROR* vid körning av `examples/wait_test` eller `examples/longrun`. När `wait` fungerar korrekt kan du även börja använda Pintos egna tester. Testet `tests/userprog/wait-simple` kan då användas som ett bra förstasteg (du kan köra det med kommandot `pintos-single-test <testnamn>`, se Pintos-Wiki för mer information).

Notera: programmet `examples/longrun_nowait` kan fortfarande ge *ERROR*. Varför?

Notera: Pintos egna tester ställer tre viktiga krav på din implementation:

1. Det får *inte* förekomma några debug-utskrifter som *inte* startar med fyrkant+mellanslag (`"# "`).
2. Implementationen av wait *måste* fungera korrekt.

3. När en process avslutar *måste* den skriva ut sin *exit_status* (parametern *status* till systemanropet *exit* om processen avslutar via ett systemanrop, -1 om processen avslutas av kernel till följa av något fel.) Den `printf` som behövs finns i `process_cleanup` men du måste *se till att variabeln status har rätt värde innan utskrivten sker.*

   ```
   printf("%s: exit(%d)\n", thread_name(), status);
   ```

Se avsnittet ”Automatiska tester” i Pintos-Wiki för information om de automatiska testerna. I slutet av kursen ska alla automatiska tester fungera.