

Systemmanagement und Sicherheit

5. Übung

Aufgabe 1 (fork-exec)

Entwickeln Sie ein C-Programm *start*, das beim Aufruf

```
start prog arg1 arg2 arg3 ...
```

zunächst ein *fork()* Aufruf ausführt und dann im Sohnprozeß das Programm *prog* via *execvp* mit den angegebenen Argumenten startet. Das Programm *prog* soll mit niedrigster Priorität ausgestattet werden, siehe Systemcall *setpriority()*

Der Vaterprozess soll weiterhin folgendes tun:

- Ausgabe der PID des gestarteten Prozesses *prog*,
- Ausgabe des Return-Codes von *prog* nach dessen Beendigung,
Hinweis: Siehe Macros unter *wait(2)*
- Ausgabe eines evtl. Signals (numerisch und eine Beschreibung des Signals),
das zum Abbruch von *prog* führte (siehe auch *psignal(3)*).

Die Deklaration von *main()* in *start.c* sei

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

Hier ist ein richtiger Aufruf von *execvp()* dabei:

- a) `execvp(argv[1][0], argv[1])`
- b) `execvp(argc, argv)`
- c) `execvp(argv[1], argv[2])`
- d) `execvp(argv[1], argv+1)`
- e) `execvp(*argv[1], *argv[1])`
- f) `execvp(**argv, **argv[1])`
- g) `execvp(argv[1], argv[1])`

Aufgabe 2 (Semaphoren und Shared Memory)

Implementieren Sie folgendes Erzeuger–Verbraucher–Schema mit Hilfe von Shared Memory und Semaphoren.

Der Erzeugerprozess P_1 hält ein Array gefüllt mit `int`-Daten, deren Anzahl sei durch eine

```
#define N_DATA 2000000
```

Direktive festgelegt. Die Daten werden von P_1 zufällig erzeugt (siehe `srand48()`, `lrand48()`).

Der Verbraucherprozess P_2 soll diese Daten erhalten, indem diese über einen von P_1 und P_2 genutzten shared memory Block übertragen werden. Im shared-memory Bereich finden weniger als `N_DATA` viele Zahlen Platz, etwa

```
#define N_SHARED 2000
```

Prozess P_1 muss also die größere Anzahl Daten in mehreren Durchläufen durch den kleineren Shared-Puffer übertragen.

Hinweise:

- die Semaphoren- und Shared-Memory-Funktionen werden erst in der Vorlesung am 3.6. besprochen. Wer schon damit experimentieren will, liest die Manualpages von

```
semget(2), semop(2), semctl(2),  
shmget(2), shmat(2), shmctl(2), shmdt(2)
```

- es empfiehlt sich ein schrittweises Vorgehen
 - zunächst eine Lösung ohne Semaphoren und nur einen Schreib-/Lesevorgang im Shared-Memory-Bereich:
 P_2 wartet mittels `sleep()`, damit P_1 Zeit hat, die Daten zu schreiben
 - danach Semaphoren hinzunehmen
 - danach mehrere Schreib-/Lesevorgänge