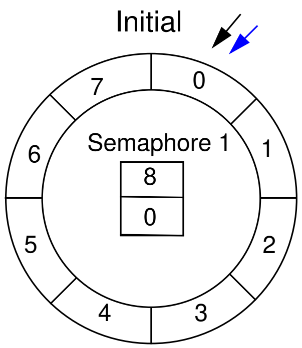
M 11. Übung

M.1 Überblick über die 11. Übung

- Besprechung 8. Aufgabe (Shared-Memory und Semaphore)
- Musterlösung zur trsh
- Klausur

M.2 Besprechung 8. Aufgabe (Erzeuger)



Verwaltungsdatenstruktur

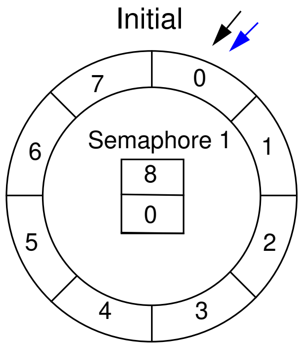
-1	PID Verbraucher
0	Leseindex
0	Schreibindex
8	Puffergröße

Semaphore 2

1

- Semaphore 1 und 2 können auch zu einer Semaphore zusammengefaßt werden
- Semaphoren erst erzeugen nachdem die Verwaltungsdaten initialisiert wurden

1 Besprechung 8. Aufgabe (Verbraucher)



Verwaltungsdatenstruktur

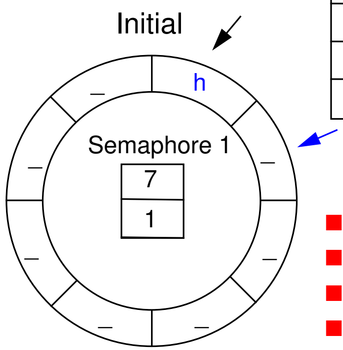
42	PID Verbraucher
0	Leseindex
0	Schreibindex
8	Puffergröße

Semaphore 2

1

- SharedMemory attachen
- Semaphoren attachen
- Semaphore 2 erniedrigen
- PID testen und schreiben
- Semaphore 2 erhöhen

2 Besprechung 8. Aufgabe (Schreiben)



Verwaltungsdatenstruktur

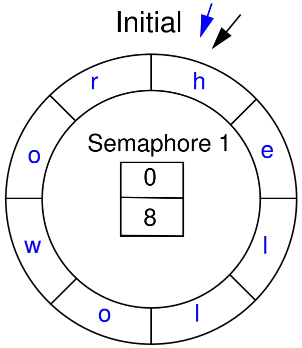
42	PID Verbraucher
0	Leseindex
1	Schreibindex
8	Puffergröße

Semaphore 2

1

- Semaphore 1 Feld 1 erniedrigen
- Zeichen schreiben
- Schreibindex erhöhen
- Semaphore 1 Feld 2 erhöhen

3 Besprechung 8. Aufgabe (Schreiben)



Verwaltungsdatenstruktur

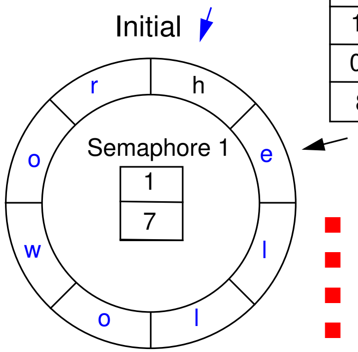
42	PID Verbraucher
0	Leseindex
0	Schreibindex
8	Puffergröße

Semaphore 2

1

- Semaphore 1 Feld 1 erniedrigen
- Zeichen schreiben
- Schreibindex erhöhen
- Semaphore 1 Feld 2 erhöhen

4 Besprechung 8. Aufgabe (Lesen)



Verwaltungsdatenstruktur

42	PID Verbraucher
1	Leseindex
0	Schreibindex
8	Puffergröße

Semaphore 2

1

- Semaphore 1 Feld 2 erniedrigen
- Zeichen lesen
- Leseindex erhöhen
- Semaphore 1 Feld 1 erhöhen

M.3 Musterlösung zur Aufgabe 5 (trsh)

- Hintergrundprozesse (Teilaufgabe b und c)
- Listenoperationen (Teilaufgabe f)
- Zeiterfassung (Teilaufgabe g)

1 Hintergrundprozesse

- yash:

```
void execute(char *commandLine, char *command, char **argv) {
  int   statloc;
  pid_t pid, ret;
  switch(pid=fork()) {
    case -1 : perror("fork failed");return;
    case 0  :
      execvp(command, argv);
      perror(command);
      exit(EXIT_FAILURE);
    default :
      while(((ret = wait(&statloc)) != pid)
          && (errno == EINTR));
       if(ret != pid)
         perror("wait failed");
       else if(WIFEXITED(statloc))
         printStatus (commandLine, WEXITSTATUS(statloc));
  }
}
```

2 Hintergrundprozesse

- Anforderungen:
  - Shell soll nicht auf Hintergrundprozess warten
  - bei einem Vordergrundprozess muss die Shell auf den richtigen Prozess warten
- mögliche Lösungen:
  - ❑ waitpid im Vaterprozess
    - waitpid kann von SIGCHLD unterbrochen werden
    - kein wait im SIGCHLD-Handler möglich
  - ❑ waitpid im SIGCHLD-Handler

3 Hintergrundprozesse

```
void execute_fg(char *commandLine,char *command,char **argv){
    switch(fg_pid=fork()) {
    case -1 : perror("fork failed"); return;
    case 0  : execvp(command, argv); /* ... */  exit(-1);
    default :
     block_all_signals(&sigmask);
     while (fg_pid!=0) sigsuspend&(sigmask);
     restore_signals(&sigmask);
     printStatus (commandLine, WEXITSTATUS(fg_status));
}  }
```

```
void sigchild_handler(int signo) {
  int status, errnobak = errno;
  while ((pid=waitpid(-1,&status,WNOHANG))>0) {
    if (!WIFEXITED(status)) continue;
    if (pid==fg_pid) {
      fg_pid=0;
      fg_status=status;
  } }
  errno = errnobak;
}
```


4 Listenoperationen

- Liste der aktiven Kindprozesse um bei SIGINT ein SIGKILL zuzustellen
- Einfügen in Liste kann durch SIGCHLD unterbrochen werden
  - Problem, wenn im SIGCHLD Handler ebenfalls Listenoperationen untergebracht sind
  - Alternativ wird das Listenelement im SIGCHLD-Handler nur markiert und im “Hauptprogramm” ausgetragen
- Einfügen muss vor Austragen/Markieren geschehen (“atomar” mit fork)

5 Listenoperationen

```
void execute_bg(char *commandLine,char *command,char **argv){
  pid_t pid;
  sigset_t sigmask;
  block_all_signals(&sigmask);
  switch(pid=fork()) {
    case -1 : perror("fork failed");return;
    case 0  :
      restore_signals(&sigmask);
      block_signal(SIGINT);
      execvp(command, argv);
      perror(command);
      exit(EXIT_FAILURE);
    default :
      if (jl_insert(pid, command)) perror ("jl_insert");
  }
   restore_signals(&sigmask);
 }
```

6 Zeiterfassung

- times liefert die verbrauchten Zeiteinheiten des aktuellen Prozesses
- und die verbrauchte Zeit seiner Kinder
- die Zeitinformationen eines Kindes werden erst durch ein wait zum Vater übertragen

7 Zeiterfassung

```
void sigchild_handler(int signo) {
  int pid,status;
  struct tms t1;
  struct tms t2;
  clock_t ut,st;
  /* ... */
  if (times(&t1)==(clock_t)-1) perror("times");

  while ((pid=waitpid(-1,&status,WNOHANG))>0) {

    if (times(&t2)==(clock_t)-1) perror("times");
    ut = t2.tms_cutime - t1.tms_cutime;
    st = t2.tms_cstime - t1.tms_cstime;
    t1 = t2;
    /* ... */

  }
}
```