Hausaufgabe 4 - Theorie

Gruppe A

Dora Szücs
Kevin Bock
Philipp Kückes
Sarah Köhler

4.1

a)

Konzept A ist Nebenläufigkeit, Konzept B ist Parallelität.

b)

Nebenläufige Prozesse (Konzept A) werden nur scheinbar gleichzeitig ausgeführt. In der Realität wechselt der Prozessor ständig zwischen den Prozessen, es wird aber immer nur ein Prozess bearbeitet. Parallele Prozesse (Konzept B) werden tatsächlich gleichzeitig ausgeführt. Voraussetzungen dafür ist, dass mehrere Prozessor-Kerne (bzw. Prozessoren) vorhanden sind, damit verschiedene Prozesse auf die Kerne verteilt und somit zur selben Zeit bearbeitet werden können. Werden nebenläufig implementierte Prozesse auf einem System mit mehreren Kernen ausgeführt, sind sie parallel.

c)

Koordination ist notwendig wenn die beiden Prozesse von einander abhängig sind, also Nachrichten austauschen müssen. Ebenso ist Koordination notwendig, wenn Prozesse die selben Ressourcen benötigen und der Zugriff geregelt werden muss. Es spielt dabei keine Rolle, ob die Prozesse nebenläufig oder parallel ablaufen.

4.2

a)

Ja, es kann zu einer Race-Condition beim Schreibzugriff auf die Datei kommen. Nach jedem geschriebenen Buchstaben kann der laufende Thread unterbrochen werden und der zweite Thread in die Datei schreiben. Deswegen könnte es folgende Ausgabe in der Datei geben:

N J A E I N N J E I A N

b)

```
Lock_Object file_lock;

T1:
while(1) {
    lock(file_lock);
      fprintf(f,"J");
      fprintf(f,"A");
    unlock(file_lock);
}


T2:
while(1){
     lock(file_lock);
      fprintf(f,"N");
      fprintf(f,"E");
      fprintf(f,"I");
      fprintf(f,"N");
    unlock(file_lock);
}
```

Es kann immer noch zu einer Race-Condition kommen, da nach jedem Freigeben der Sperre (file_lock) beide Threads die Sperre neu setzen könnten. Welcher zuerst die Sperre setzt, hängt also davon ab, welcher Thread als nächster aktiv wird. Somit können nun Ausgaben entstehen, wo zwar alle Worte je vollständig sind, allerdings nicht sichergestellt ist, dass ja und nein abwechseln geschrieben werden. Beispielsweise:
J A J A N E I N N E I N J A J A

c)

```
Lock_Object file_lock;
signal_Object JA_fertig;
signal_Object NEIN_fertig;


NEIN_fertig = true;

T1:while(1){
    wait(NEIN_fertig)
    lock (file_lock)
     fprintf(f,"J");
     fprintf(f,"A");
     signal(JA_fertig)
    unlock(file_lock);
}
```

```
T2:
while(1){
    wait(JA_fertig)
    lock (file_lock)
      fprintf(f,"N");
      fprintf(f,"E");
      fprintf(f,"I");
      fprintf(f,"N");
      signal(NEIN_fertig)
    unlock(file_lock);
}
```

d)

In der Lösung von c muss immer abwechselnd JA und NEIN in die Datei geschrieben werden, da keine Race Condition mehr entstehen kann. Dagegen ist bei 4b nicht sichergestellt, dass die Threads abwechselnd in die Datei schreiben, d.h. es könnte auch mehrfach das selbe Wort direkt aufeinander folgen.

Aufgabe 4.3

Spurious Wakeups bezeichnet das Aufwachen eines Threads ohne dass das Signal, auf welches er wartet, ausgelöst wurde. Das heißt, er scheint ein Signal zu empfangen, obwohl keines gesendet wurde und die Bedingung dafür nicht erfüllt ist.

Die Lösung aus 2c berücksichtigt das Problem der Spurious Wakeups nicht, da nach dem Empfangen eines Signals die zugehörige Bedingung nicht mehr geprüft wird. Eine Lösung, welche auch Spurious Wakeups berücksichtigt ist folgende:

```
Lock_Object file_lock;
signal_Object JA_fertig;
signal_Object NEIN_fertig;
int last_word = 0;

NEIN_fertig = true;

T1:while(1){
  while ( last_word == 1){
    wait(NEIN_fertig)}
    lock (file_lock);
     fprintf(f,"J");
     fprintf(f,"A");
     last_word = 1;
     signal(JA_fertig);
    unlock(file_lock);
}
```

```
T2:
while(1){
 while ( last_word == 0){
    wait(JA_fertig)}
    lock (file_lock);
      fprintf(f,"N");
      fprintf(f,"E");
      fprintf(f,"I");
      fprintf(f,"N");
      last_word = 0;
      signal(NEIN_fertig)
    unlock(file_lock);
}
```