https://folk.ntnu.no/michaeng/tdt4186_21/
michael.engel@ntnu.no

Theoretical exercises
Spring 2021

Theoretical Exercises 3

Synchronization

Please submit solutions on Blackboard by Thursday, 25.02.2021 12:00h

3.1 Resource allocation graphs (3 points)

Consider a system with four processes $P1 \ldots P4$ which want to access five exclusive, non preemptible resources $R1 \ldots R5$.
The atomic requests for the resources are arriving in the following order:
P1 $\rightarrow$ R3, P3 $\rightarrow$ R1, P4 $\rightarrow$ R2, P1 $\rightarrow$ R5, P3 $\rightarrow$ R3, P4 $\rightarrow$ R5, P2 $\rightarrow$ R4 and finally P1 $\rightarrow$ R1.

a. Draw the resource allocation graph that results from this sequence of resource requests.

b. Which condition has to be fulfilled for a deadlock to occur?

c. Is there a deadlock present in the system described above? Explain your answer.

3.2 Semaphores (4 points)

We have three programs Pa, Pb and Pc. Pa contains the functions $a1()$, $a2()$ and $a3()$, Pb has the functions $b1()$ and $b2()$ and finally Pc has the functions $c1()$, $c2()$ and $c3()$. When one of these functions is executed, it simply prints its own function name.
In addition to the functions described, each of the programs also contains a main() function which calls the respective functions of the program one after the other (Pa: first $a1()$, then $a2()$ and finally $a3()$, Pb: $b1()$ then $b2()$, Pc: $c1()$, then $c2()$ and finally $c3()$). We also have three semaphores Sa, Sb and Sc.
Using one-sided (unilateral) synchronzation, synchronize the processes (running in parallel) belonging to the three programs described above so that the output on the screen reads as follows:
`a1 b1 a2 c1 c2 b2 a3 c3`
All functions have the same structure, e.g. for $a1()$:

```
void a1() {
  <possibly block here using wait()>
  printf("a1 ");
  <possibly signal here using signal()>
}
```

a. To which *initial values* do you have to set the semaphores Sa, Sb and Sc?

b. Fill in a table that indicates the *required* calls to the semaphore functions wait() and signal() in the respective functions of Pa, Pb and Pc, giving the used semaphore as parameter to the call, e.g. `wait(Sb)` or `signal(Sc)`. If no synchronization is required, enter "–" in the table cell.

	a1	a2	a3	b1	b2	c1	c2	c3
wait(...)								
signal(...)								

3.3 Even more semaphores (1 point)

How many times does the following short C program print the letter X? Assume that the semaphore `sem` is initialized to the value 4.

```
int main(void) {
  for ( ; ; ) {
    printf("X\n");
    wait(&sem);
  }
  printf("X\n");
  return 0;
}
```

3.4 Synchronization using interrupts (2 points)

On x86 CPUs, interrupts can be disabled and reenabled using the machine instructions `cli` and `sti`. Why is this a significant problem (and, as a consequence, not allowed to be performed by regular user programs)?