# Exam's Model Answer of: Operating Systems II

## Questions

1- Reader Writer model (1)

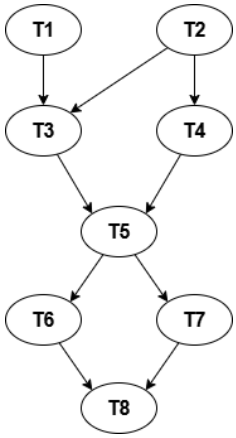
2- $kill\ -cont\ <pid>$ (1)

3- Sockets (1)

## Exercise 1:

I. $S = \{T1, T2, T3, T4, T5, T6, T7, T8\}$

1. Precedence graph: (1)




2. Parallel program: (1.5)

```
ParBegin
    Begin
        T1
        Wait(T2)
        T3
    End
    Begin
        T2
        Signal(T3)
        T4
    End
ParEnd
T5
ParBegin
    T6
    T7
ParEnd
T8
```

3. To verify the maximal parallelism of $S$, we must verify (1) its determination and (2) all arcs are useful.

   Determination:

   Parallel tasks are: $T1//T2, T3//T4, T1//T4, T6//T7$.

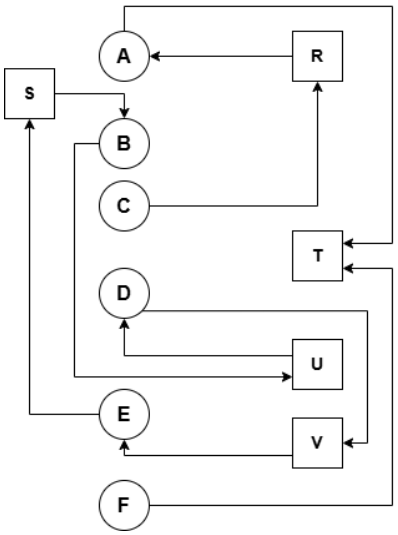
   $T1//T2$: $R1 \cap W2 = W1 \cap R2 = W1 \cap W2 = \emptyset$ (verified).

   $T3//T4$: $R3 \cap W4 = \{b\} \neq \emptyset$.

   $S$ is not determined, therefore $S$ is not in maximal parallelism. (1.5)

II. Deadlock:

1. Resource Allocation Graph (RAG) (2)




2. There is cycle: $B \rightarrow U \rightarrow D \rightarrow V \rightarrow E \rightarrow S \rightarrow B$

   Therefore, the is a deadlock. The processes involved are: B, D, E. (2)

## Exercise 2:

Only the code parts related to the critical resources must be synchronized.

In this exercise $P1$ and $P2$ share only one variable ($x$).

1. Using the Peterson Algorithm: (2.5)

   $lock: 0 \rightarrow used$

   $1 \rightarrow free$

   Initialization: $lock = 1$

| $P1$ | $P2$ |
|---|---|
| $While(lock == 0);$<br>$lock = 0;$ //critical section<br>$Read(x)$ | $Read(a)$<br>$While(lock == 0);$<br>$lock = 0;$ |

| | |
|---|---|
| $y = 2x$<br>$lock = 1$; //Outside the critical section<br>$Print(y)$ | $x = a^2$<br>$lock = 1$; //Outside the critical section<br>$z = 2a$ |

2. Using semaphores: (2.5) A binary semaphore $sem\_cs$ (only one process can use $x$ at a time).

Initialization: $sem\_cs$

| $P1$ | $P2$ |
|---|---|
| $wait(sem\_cs);$<br>$Read(x)$<br>$y = 2x$<br>$signal(sem\_cs);$<br>$Print(y)$ | $Read(a)$<br>$wait(sem\_cs);$<br>$x = a^2$<br>$signal(sem\_cs);$<br>$z = 2a$ |

## Exercise 3: (4)

```
pipe_fd = create_unnamed_pipe() // [read_end, write_end]

if fork() == CHILD:
   close(pipe_fd.write_end)
   data = read_all(pipe_fd.read_end)
   close(pipe_fd.read_end)
   print(transform(data))
   exit(0)
else:
   close(pipe_fd.read_end)
   write(pipe_fd.write_end, payload)
   close(pipe_fd.write_end)
   wait_child()
```