Kandidatnummer: 463

Eksamen - Kanditatnr: 463

1)

There is ( ) where Im not sure of what the question mean. Like j). It is True if we dont want deadlock and its False if we want the possibility of deadlock..

a) False

b) True

c) False (the stack grows)

d) True

e) True

f) True

g) True

h) False (it causes problems when 2 writes simultaneously)

i) True

j) False (it has to request more resources to get deadlock)

2)

a) A resource deadlock happends if each process enters a waiting state because a recource is held by another process, which is waiting for another.. in a loop. A recource can be printers, tape drivers, of piece of information like files and data records. Resources can only be used by one process at the time. When a deadlock happends, the system is stuck because the processes are waiting for each other.

b)

- Mutial exclusion: One or more resources must be held by a process. If the process doesnt hold a process is also doesnt prevent others from using the resource
- Hold and wait: One or more resources must be held by a process and at the same time the process must request even more resources which is held by another process
- Non-preemtion: A resource that is being held can only be released by the process which is holding it
- Circular wait: All the processes is waiting for a resource which another process holds.. it goes in a loop.. to which at the end whre the last process is waiting for the first process to realease its resource

c)
All the conditions have to be met to have the risk/posibility of getting a deadlock.

3)

1. Not recently used
2. First in, first out
3. Second chance
4. Clock (LRU)
5. Least recently used
6. Aging
7. Not frequently used
8. Longest distance first

4)

a)

Owner = Rw- = 6

= 600

b)

Group = R-x = 5

= 050

c)

Others = — = 0

= 000

(All added together = 650)

5)

a)

The output should be first “Creating Thread 0 in main() function” then “Hello from Thread 0”. After that ”Creating Thread 1 in main() function” then “Hello from Thread 1”.
Whats happening now is that main() for-loop is so fast that is creates 2 threads before the threads gets do do their job. But, if you make the for-loop wait in any way, it will let 1 thread be created, then make it do its job before creating a new one.

b)

After line 20 it should be placed a phread_join(thread[I]). This will make the main wait for the thread before it creates a new one.

It could also be done by having a bool that if its true, it can run again. If its false it has to wait. The first thread can set the bool value to true when its done..

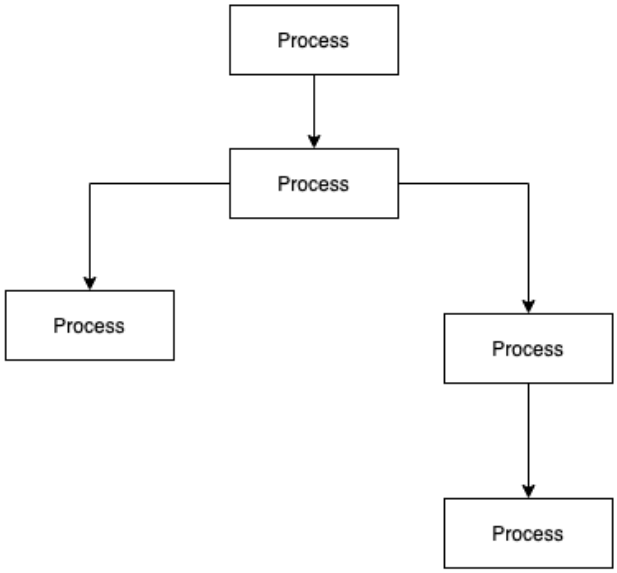
6)

```c
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>

sem_t sem;
void *Thread1Print(void *ThreadId) {
    printf("Hello");
    sem_post(&sem);
}
void *Thread2Print(void *ThreadID) {
    sem_wait(&sem);
    printf("Goodbye");
    sem_post(&sem);
}


int main() {
    int i;
    pthread_t td1, td2;
    sem_init(&sem, 0, 0);

    for(i = 0; i > 10; i++) {
        pthread_create(&td1, NULL, Thread1Print, NULL);
        pthread_create(&td2, NULL, Thread2Print, NULL);
        pthread_join(td1, NULL);
        pthread_join(td2, NULL);
        sleep(3);
    }

    printf("See you agian");


    sem_destroy(&sem);
}
```

Because I run on a Mac and POSIX unamed semaphores are not implemented on Mac I cant see if it runs (though I have been told that it doesnt matter because you already know our skills).

But, to be clear is that creating thread from the same phread again (td1..) Im not entirely sure works. If it doesnt work it could be made with td[n] and have a n amount of pthreads if that made any sence .. instead of creating a thread using the same td variable.

7)



8)

a) NRU would pick page 0 beacuse its sorted and it picks at random the first numbered page.
b) FIFO would pick the first loaded. So that means page 1.

9)

a) 1000 0000 0000 0000 —> 1111 1100 0000 0000
b) 1111 1100 0000 0000 —> 1111 1111 0000 0000
c) 1111 1111 0000 0000 —> 1111 1111 1100 0000
d) 1111 1111 1100 0000 —> 1111 1111 1111 0000
e) 1111 1111 1111 0000 —> 1000 0011 1111 0000

10)

a)

	RS1	RS2	RS3
PA	1	1	0
PB	0	0	1
PC	0	1	0

	RS1	RS2	RS3
PD	1	0	0

This is not a deadlock becuse RS3 has still one more instance we can use from it. And process PA only needs 1 instance of RS3 to finish. So to proceed:

1. 1 RS3 to PA. Then PA is finished and releases its resources
2. Then PB gets 1 RS1 and 1 RS2. . Then PB is finished and releases its resources
3. Then PC gets 1 RS1 and 1 RS3. . Then PC is finished and releases its resources
4. Then PD gets 1 RS2 and 1 RS3. . Then PD is finished and releases its resources
5. Now its solved

b)

	RS1	RS2	RS3
PA	1	1	0
PB	0	1	1
PC	1	0	0
PD	0	0	1

This is a situation is a deadlock. So, we can recover with preemtion to solve the problem. Take 1 from PA´s RS1. Then give it to PB. Now its not a deadlock anymore.

11)
When the system performes a rollback it

1. Checks what resources is needed at what state
2. Then it will modify the resource ordering/allocation to avoid the deadlock
3. Also the processes that posess the resources will have to wait

But anyways. To now make sure it wont deadlock again the system will have to find a way to execute a path without entering the unsafe state.
(Unsafe state is when the system cant be sure of all the processes will finish
Safe state is when the system is sure that all the processes wil finish). So to find the safe path it uses algorithms. It can use:

1. The bankers resource (Single resource)
2. The bankers resource (Multiple resources)

Say that the OS use one of these algorithms. If it uses (Single resource):

- It first checks that the path is safe.
- If it is, the request is carried out. But if its not, the request will be denied

If it uses (Multiple resources)

- Find a row where the unmet resources is lower/equal that of A
- Then it gives the resource it needs so it finishes and releases is resources
- Then because it has more resources now it can do it step 1 and 2 until all the processes has been terminated

This is how the system rolls back, and avoids deadlock