School of Engineering and Applied Science

CSE332 - Operating Systems

End Semester Exam

Name - Darsh Patel
Roll No - AU1940150

Que 1
(a)

Description:

(b)
Description:
We declared a variable named thread id in main(), which is of type pthread t and is an integer that is used to identify the thread in the system. We used the pthread create() function to create three threads after defining three thread ids.

In the interim, we've used mutex to synchronise all of the threads.

We have a lot of processes in an Operating System, and these processes require a number of resources. Three threads are utilising the same variable "num" in this case. They read the variable, then update the value of the variable, and then write the data to memory.

You can see from the above that a process will have to read the value of num by 1 after doing various operations.There are three threads that need to be run right now.
Now, to synchronise all three threads, we used the mutex lock in the task function to ensure that no more than one thread is utilising the value of num at the same time, and then we printed the num's ascii value and integer value. The mutex was then unlocked, allowing other threads to make changes as well.

Que 2
(a)

Description:
The round-robin (RR) scheduling algorithm is mostly utilized for time sharing systems. It is almost the same as FCFS scheduling, but here we enable the system to switch between processes (preemption). We define a small unit of time. The queue that is ready is treated as a circular queue. The scheduler in the CPU goes around the ready queue and each process is allocated to the CPU for some unit of time (1 time quantum).

(b)
Description:

The buddy system is implemented in the following manner: At all times, a list of free nodes of all possible powers of 2 is maintained (so if total memory size is 1 MB, we'd have 20 free lists to track-one for blocks of size 1 byte, one for blocks of size 2 bytes, one for blocks of size 4 bytes, and so on).
When we receive a request for allocation, we look for the smallest block that is larger. If a suitable block is located on the free list (for example, if the request is 27 KB and the free list tracking 32 KB blocks contains at least one element), the allocation is completed; otherwise, we traverse the free list higher until we find a suitable block. Then we separate it into two blocks, one for adding to the next free list and the other for deleting (of smaller size), one to descend the tree until we reach the goal and return the user's requested memory block We just return null if no such allocation is possible.

Que 3
LOGIC

```
Mutex mutex; // Used to provide mutual exclusion for critical section
Semaphore empty = N; // Number of empty slots in buffer
Semaphore full = 0 // Number of slots filled
int in = 0; //index at which producer will put the next data
int out = 0; // index from which the consumer will consume next data
int buffer[N];
```

Producer Code :

```
while(True) {
 // produce an item
 wait(empty); // wait/sleep when there are no empty slots
 wait(mutex);
 buffer[in] = item
 in = (in+1)%buffersize;
 signal(mutex);
 signal(full); // Signal/wake to consumer that buffer has some data and they can consume now }
```

Consumer Code

```
while(True) {
 wait(full); // wait/sleep when there are no full slots
 wait(mutex);
 item = buffer[out];
 out = (out+1)%buffersize;
 signal(mutex);
 signal(empty); // Signal/wake the producer that buffer slots are emptied and they can produce
more
 //consumer the item
}
```

Code :