Final

CIS 415: Operating Systems

Fall xxxx

dd December xxxx, 12:30-14:30 (120 minutes)

NAME: _____

Section	Total Points	Points Scored
1. File systems	20	
2. Pthreads and synchronization	20	
3. Real time scheduling	20	
4. Memory	20	
5. Synchronization and deadlock	20	
6. Miscellaneous	20	
Total	120	

Comments:

1. Take a deep breath.
2. Write your name and initials on this page NOW and initial all other pages when you are told to start the exam.
3. Do all problems. Read each question ***carefully*** to be sure you are answering the question being posed.
4. Questions are intended to be answered succinctly. Concise and direct is better than rambling.
5. If you have a question, raise your hand and we will come to you, if we can. Otherwise, we will acknowledge you and you can come to us.

Your initials _____

1. File systems

a) Consider a file system implemented over a 100GB[1] disk, using 1024 byte blocks. Describe how such a file system would be constructed using:

Need to compute the number of blocks on the disk, independent of the allocation model: 100GB/1024 = 104,857,600 blocks; from this they reason that each mapping entry in an index block or the link in linked allocation requires 32 bits [1]. Also need to indicate the necessity to maintain a file header for each file in each of the allocation models [1].

(i) contiguous allocation [2]

Need to discuss requirement that starting block number and size in bytes appears in the file header [1].

(ii) linked allocation [3]

Need to indicate that starting block number and size in bytes appears in the file header[1]. 1st four bytes of each data block reserved for block number of next block in the file[1].

(iii) indexed allocation [3]

Need to indicate that 1st index block number and file size appears in the header [1], and that index blocks are allocated contiguously to address large files[1]. [1] mark reserved for lucid discussion of indirect block usage.

b) For each of these allocation methods, estimate how long it would take to access the 1,000,000th byte of a file, assuming that the relevant directory was present in memory but that the file itself had never been accessed. Justify your answer, stating any assumptions and explaining all of the numbers you choose to use in your calculations. [8]

[1] for working out that they need a block in the high 900's (10^6/1024 = 976.5625).
[1] for how long a disk access takes (probably expect around 10ms worst case, or 5ms on average, depending on numbers).
[1] for indicating 1 disk access to fetch header/inode regardless of allocation model.
(i) fetch it in 1 step costing a disk access [1].
(ii) only 1020 usable bytes per block, so need 981st block; must incur 981 disk accesses
(iii) they will have to have concluded that only 256 direct blocks are pointed to in each index block [1], so they will have to obtain the block number from the 4th index block, thus requiring 2 disk accesses, one to fetch the 4th index block, the other to fetch the block containing the byte [1].

c) The Linux command "ls –R", which recursively lists directory contents, does not normally follow symbolic links. Why not? What changes would be needed to the ls program and/or the file system to allow links to be followed by default? [4]

[Discussed during lectures. 1 mark per relevant correct point made. Max of 2 marks for explanation of problem, max of 3 marks for solution, but max 4 overall.]

Problem: can use symbolic links to build a cycle, recursively following that in naïve implementation will give infinite amount of output.

Solution: must either mark the file system as we're walking it (but that restricts us to one ls at a time) or keep track in ls of visited directories and watch for re-visiting (which is more expensive)

[1] 100 GB = 107,374,182,400 bytes

Your initials _____

2. PThreads and synchronization

a) How does PThreads support the ability for threads to create critical regions around shared data? How does it support the creation of conditional critical regions around shared data? [6]

PThreads provides the abstraction of a mutex for creating critical regions; a mutex is used to achieve mutually exclusive access to shared data; each thread that wants to access data protected by a mutex must perform a lock(mutex)/unlock(mutex) sandwich around code that accesses the data.

To support conditional critical regions, PThreads provides the additional abstraction of a condition variable; a condition variable, when used together with an associated mutex, enables one thread to signal that an event has occurred that might be of interest to other threads; enables one to implement conditional critical regions based upon values of the shared state. after locking the associated mutex, code can test the shared state, and wait on the condition variable if it is not in the correct state to proceed.

For each type of critical region, 1 mark for name of mechanism, 2 marks for adequate description of abstraction and use.

b) The counting semaphore abstraction combines mutually exclusive access to shared data with conditional synchronization. As a reminder, a counting semaphore is created with an initial value, create(int N), where N > 0; two atomic operations, called wait() and signal(), are supported on a counting semaphore, and logically do the following:

```
void wait(semaphore S) {
   while (S <= 0)
      ;    /* do nothing */
   S--;
}

void signal(semaphore S) {
   S++;
}
```

Sketch a C implementation for a counting semaphore ADT satisfying the following interface definition using the PThreads support you have described in your answer to 4(a) above: [14]

```
typedef struct semaphore Semaphore;

Semaphore *sem_create(int N);
void sem_wait(Semaphore *s);
void sem_signal(Semaphore *s);
```


Your initials _____

(blank page for your code sketch)

```
#include <stdlib.h>
#include <pthread.h>

typedef struct semaphore Semaphore;

struct semaphore {
    int count;
    int size;
    pthread_mutex_t lock;
    pthread_cond_t cond;
};

Semaphore *sem_create(int N) {
   Semaphore *sem = (Semaphore *)malloc(sizeof(Semaphore));

   if (sem != NULL) {
      sem->count = N;
      sem->size = N;
      pthread_mutex_init(&(sem->lock), NULL);
      pthread_cond_init(&(sem->cond), NULL);
   }
   return sem;
}

void sem_signal(Semaphore *sem) {
    pthread_mutex_lock(&(sem->lock));
    if (sem->count < sem->size)
        sem->count++;
    pthread_cond_signal(&(sem->cond));
    pthread_mutex_unlock(&(sem->lock));
}

void sem_wait(Semaphore *sem) {
    pthread_mutex_lock(&(sem->lock));
    while (sem->count <= 0)
        pthread_cond_wait(&(sem->cond), &(sem->lock));
    sem->count--;
    pthread_cond_signal(&(sem->cond));
    pthread_mutex_unlock(&(sem->lock));
}
```

1 mark for #includes, 4 marks for appropriate structure, 3 marks for create [malloc, check malloc return, initialization of elements], 3 marks for signal [1 for lock/unlock sandwich, 1 for update of count, 1 for signal], 3 marks for wait [1 for lock/unlock sandwich, 1 for while test/wait() call, 1 for signal]

Your initials _____

3. Real-time scheduling

a) Least Slack Time first (LST) is a well-known dynamic-priority scheduling algorithm for periodic tasks. Describe how LST works, and why it is classified as a dynamic-priority algorithm. [4]

LST dynamically assigns priorities according to the slack time for the job – i.e. the difference between the deadline minus the current time minus the execution time needed for the job. The smaller the slack time, the higher the priority. Thus, the priority for a job changes as we get closer to its deadline.

LST is an optimal scheduling algorithm, such that if the total utilization is 1.0 or less, it yields a feasible schedule.

b) There are two variants for LST, non-strict and strict. Describe the difference between these variants. Which is variant is commonly used? [4]

non-strict LST computes the slack time for jobs in the queue whenever the current job completes or another job is released; strict LST continuously monitors the slack time for each job, making a scheduling decision whenever the slack time of one of the jobs is smaller than the current job. Given the high overhead of strict LST, it is never used.

c) You are given a set of independent, periodic tasks: $T_1 = (0, 2, 1)$, $T_2 = (0, 10, 3)$, $T_3 = (0, 20, 3)$.

(i) Is there a feasible schedule for this set of tasks using non-strict LST? Why or why not? [4]

The total utilization is $1/2 + 3/10 + 3/20 = 0.95$; since LST is an optimal scheduling algorithm, this means that there is a feasible schedule for this set of tasks.

(ii) Assume that you are asked to add another task, $T_4 = (0, \pi, 1)$. What is the minimum value of π such that this new system, $\{T_1, T_2, T_3, T_4\}$ has a feasible schedule using non-strict LST? Show your work. [8]

Since the total utilization in (i) above is 0.95, this means that there is 0.05 utilization left for T_4 ➔ $0.05 = 1 / \pi$ ➔ $\pi = 20$

Your initials _____

4. Memory

d) Consider a demand-paging system with the following time-measured utilizations:

CPU utilization	20%
Paging disk	97.7%
Other I/O devices	5%

For each of the following, indicate whether it will (or is likely to) improve CPU utilization. Explain your answers.

(i) Install a faster CPU [2]

No, will execute CPU bursts more quickly, leading to an increased rate of page faults, leading to even more paging activity.

(ii) Add prepaging to the page-fetch algorithm [2]

No, will lead to even more paging I/O.

(iii) Decrease the degree of multi-programming [2]

Yes, means that each process in memory can have more pages in its residence set.

(iv) Install more main memory [2]

Yes, means that each process in memory can have more pages in its residence set.

(v) Install a faster hard disk or multiple controllers with multiple hard disks. [2]

May improve the CPU utilization slightly, as more disk I/O can take place; will not have a major effect, since it takes so long to resolve each page fault and we have not increased the average residence set size.

e) Which of the following programming techniques and structures are "good" for a demand-paged environment? Which are "not good"? Explain your answers.

(i) Stack [2]

Good, as both push() and pop() actions exhibit locality of reference.

(ii) Hashed symbol table [2]

Not good, as hash function causes pseudo-random access to array of listheads.

(iii) Sequential search [2]

Good, sequential search exhibits locality of reference.

(iv) Binary search [2]

Not good, as binary search causes non-local accesses.

(v) Indirection [2]

Not good, as indirection leads to non-local accesses.

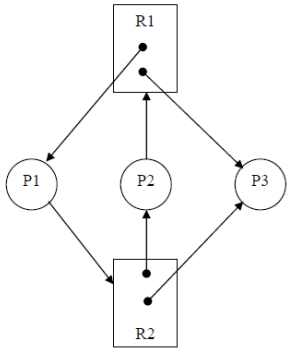
Your initials _____

5. Synchronization and Deadlock

a) What are the necessary conditions for a deadlock to occur? [4]

- Mutual exclusion: at least one resource must be held in a nonsharable mode; i.e., only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
- Hold and wait: a process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
- No preemption: resources cannot be preempted; i.e., a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- Circular wait: A set { P0, P1, …, Pn } of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, …, Pn-1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.

b) Consider the following resource allocation graph. Rectangles indicate resources; circles indicate processes. There is an arrow from a process to each resource it is requesting but does not hold, and an arrow from each resource to a process that holds a lock on that resource.



Is this system deadlocked? Why or why not? (Give a precise reason why there is a deadlock or how you can prove that the system is not deadlocked.) [3]

P3 does not satisfy the hold and wait criterion enumerated above. It holds all resources required, and may execute to completion. Once P3 terminates, it will release its resources (one each of R1 and R2). At that point, both P1 and P2 will be able to acquire the resources they are waiting for and will be able to execute.

c) Explain under what circumstances cooperating threads would require use of Conditional Critical Regions. Give an example. [2]

Critical regions guarantee serialized access by contending threads to shared data. If a thread's use of the shared data is conditional on the current state of the shared data, we need conditional critical regions.

A good example where conditional critical regions are necessary is a finite/bounded buffer queue, where the producer[s] should not add data if the buffer is full, waiting until there is room, and the consumer[s] cannot retrieve data if the buffer is empty, waiting until there is data in the buffer.

Your initials _____

d) Given the example that you have provided in part (c) above, sketch C code necessary for the cooperating threads to implement the relevant CCR using the following primitive functions: [11]

```
/* mutex functions */
Mutex m = M_INITIALIZER; /* declare and initialize */
void Lock(Mutex m);      /* lock */
void Unlock(Mutex m);    /* unlock */

/* condition variable functions */
Condition c = C_INITIALIZER;    /* declare and initialize */
void Wait(Condition c, Mutex m); /* wait until signalled */
void Signal(Condition c);        /* signal */
void Broadcast(Condition c);     /* broadcast */
```

Be sure to provide comments with your C statements to indicate what you are doing and why.

In main(): (3 for correct initialization)

```
#define NBUFFERS 10

Mutex m = M_INITIALISOR;        /* mutex to protect shared data */
Condition notEmpty = C_INITIALIZER;/* cond signalled by prod */
Condition isRoom = C_INITIALIZER;  /* cond signalled by cons */
int nFree = NBUFFERS;              /* number of free buffers */
int nFull = 0;                     /* number of full buffers */
```

In producer(): (4 for correct synchronization)

```
Lock(m);            /* enter critical section */
while (nFree <= 0)  /* cannot place data in Q until free buffer */
    Wait(isRoom, m);/* wait until isRoom is signalled */
nFree--;            /* using one buffer */
/* store data in free buffer */
nFull++;            /* indicate that one more buffer is full */
Signal(notEmpty);   /* signal that one more buffer is full */
Unlock(m)           /* leave critical section */
```

In consumer(): (4 for correct synchronization)

```
 Lock(m);           /* enter critical section */
while (nFull <= 0)  /* cannot remove data from Q until full buffer */
    Wait(notEmpty, m);/* wait until notEmpty is signalled */
nFull--;            /* taking first full buffer */
/* consume data from the next full buffer */
nFree++;            /* indicate that one more buffer is free */
Signal(isRoom);     /* signal that one more buffer is free */
Unlock(m)           /* leave critical section */
```


Your initials _____

6. Miscellaneous

a) An operating system provides a number of abstractions for use by processes over the hardware of the computer. As with any large program, a modular structure to the OS enables significantly easier tailoring of operating system function. One well-known modularity method is to break the OS into a number of layers. One such layered approach is the concept of a virtual machine.

(i) Describe the basic concepts and attributes of a virtual machine. Use a figure if it helps in your explanation. [4]

Fundamental idea is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, etc.) into several different execution environments, creating the illusion that each separate execution environment is running its own private computer. The virtual machine provides an interface that is identical to the underlying bare hardware. Enables the execution of multiple operating system instances, each with their own set of processes, on a single computer.

(ii) Compare and contrast full virtualization and paravirtualization. Give an example of each that is available in the industry. [4]

Full virtualization refers to running an unmodified OS on a virtual machine (e.g. VMWare). Paravirtualization refers to modifying the OS to run on the virtualized environment rather than on bare metal (e.g. Xen).

(iii) Use of virtualization is now commonplace. Give an example of one such common usage, and why virtualization is key to that usage. [2]

Expect them to talk about infrastructure as a service, and providers such as Amazon. Any other common usage of virtualization will be acceptable.

b) Explain how the 2-handed clock page replacement algorithm works, carefully describing the underlying implementation (e.g. what data structures are required). Discuss how the implementation might be affected by the choice of local or global page replacement. Briefly indicate how the implementation would be affected by sharing of pages between address spaces. [10]

Requires careful discussion of accessed bit (its existence[1], that the MMU sets it[1], that the page replacement algorithm clears it[1]), two pointers (next page to consider for replacement[1], next page to clear accessed bit[1]), idea of second pointer as offset from first clearing bits as hands advance[1]. Sharing needn't make any difference to actual algorithm (just increases accessed rate)[1], but may wish to avoid expelling shared pages[1]. Local vs global is just where do clock pointers live and how many are there[1]; indicate that local page replacement policies may cause large applications to thrash unnecessarily[1].

Your initials _____

Your initials _____