TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2004

MID-YEAR

COMP 305

Operating Systems

Time Allowed: 3 Hours

Instructions: Answer question one.

Answer **five** of questions two through seven.

Each question is worth 25 marks and should take you about 25 minutes.

Paper foreign language dictionaries are permitted.

Non-programmable calculators without full alphabetic keys are permitted.

Electronic dictionaries and programmable calculators are not permitted.

Question 1. Warm Up [25 marks]

A few short questions to get the mind working. Remember to allow approximately one minute for each mark.

(a) [4 marks] Give two important differences between user-level threads and kernel-supported threads.

User-level threads exist wholly within the process and are not visible to the operating system. They are very inexpensive to create, destroy, and switch amongst. However, if one blocks, the whole process blocks. Kernel-supported threads are more expensive because system calls are needed to create and destroy them and the kernel must schedule them. They are more powerful because they are independently scheduled and block individually.

(b) [6 marks] Briefly describe the four necessary conditions for deadlock?

Mutual exclusion *At least one resource must be held in a non-sharable mode. If a second process requests the resource it must wait for the first to free it.*

Hold and wait *Processes hold the resources they have already been granted while waiting for additional resources to be allocated.*

No preemption *Resources cannot be preempted. A resource may only be released voluntarily by the process that is holding it.*

Circular wait *There must be a set of processes such that each is waiting on a resource currently held by another. Since all are waiting none will release the resources they are holding.*

(c) [5 marks] Per Brinch Hansen proposed a programming language construct called conditional critical regions:

region V when B do S;

V is a shared variable and B is a boolean expression. The program waits until B is true, locks V and then executes S.

In one or two sentences explain the difficulty with this construct.

Since B may be arbitrary the thread must be woken to evaluate repeatedly evaluate B. Compare with wait and signal.

(d) [4 marks] Define *virtual memory.* Identify **two** memory management problems that are solved by the use of virtual memory.

Virtual memory is the run time binding of a logical or virtual memory address to a physical or machine address. It solves (1) allocation problems by allowing a program to be loaded anywhere and (2) sharing by allowing memory (e.g. text segments) to be mapped into multiple processes.

(e) [6 marks] Consider the logical address format below used in paged virtual memory.

Sketch the usual data structures used in such systems and outline the steps involved in binding a logical address to a physical address.

Expect to see a page table with a frame number indexed by the page number. Add the offset to arrive at the final physical address.

Question 2. Synchronization [25 marks]

Operating systems have many concurrent activities that may interfere with each other.

(a) [3 marks] Explain why the operating system of a single processor computer must worry about concurrent activities.

Although a single processor can execute only one sequence of instructions at a time, the occurrence of interrupts (from i/o devices, alarms, etc.) may occur at any time and give the appearance of arbitrary concurrency.

(b) [4 marks] State the properties that must be true of a correct solution to the critical section problem.

The critical section problem involves controlling access of multiple processes to a segment of code (the critical section) s.t.:

at most one process is in the critical section at one time,
if no process is executing in the critical section the selection of the next process cannot be postponed indefinitely, and
no process can be made to wait indefinitely.

(c) [6 marks] Consider the *swap* instruction defined by

```
void swap (bool a, b) {
    bool temp;
    temp = a;
    a = b;
    b = temp
}
```

Does the following use of *swap* successfully implement a critical section? Justify your answer.

```
lock = false;                  // global variable lock = true if held
...
key = true;                    // local variable key
while(key) swap(lock,key);
    // the critical section
lock = false;
```

No. It satisfies mutual exclusion and progress, but bounded waiting is not satisfied.

(d) [12 marks] You are asked to port Nachos to a system supporting the *swap* instruction defined above. *Outline* an implementation of a Nachos *lock* for this new machine.

```
// There is now no need to disable interrupts.
// The class global lockIsHeld indicates whether or not
// the lock is in use.  If it is false we can proceed.
// Test this by swapping TRUE into it and checking the
// result.  Otherwise, just like Nachos.

Lock::Lock(char* debugName)
{
   name = debugName;
   queue = new List;
   lockIsHeld = FALSE;
}

Lock::~Lock()
{
   delete queue;
}

void Lock::Acquire()
{
   int inUse = TRUE;
   swap(inUse,lockIsHeld);
   while (inUse) {
     queue->Append((void *)currentThread);
     currentThread->Sleep();
     swap(inUse,lockIsHeld);
   };
}

void Lock::Release()
{
   int inUse = FALSE;
   Thread *thread = (Thread *)queue->Remove();
   if (thread != NULL) { scheduler->ReadyToRun(thread); };
   swap(inUse,lockIsHeld);
}
```

Question 3. CPU Scheduling [25 marks]

The processor is a key resource that all operating systems must manage. Without access to the processor a program cannot execute.

(a) [8 marks] List and explain *four* criteria which may be used in evaluating the performance of a scheduling algorithm.

CPU utilisation, throughput, turnaround time (batch systems), waiting time and response time (interactive systems)

(b) [3 marks] Describe the optimal scheduling algorithm.

The optimal algorithm always selects the job with the shortest processing time first. Unfortunately, this requires that we know what will happen in the future and so the best we can do is approximate it.

(c) [5 marks] After the operating system allocates the processor to a user thread, how does it regain control of the processor at some point in the future? Draw on your experience with Nachos in answering the question.

The user process will either relinquish the processor voluntarily through a system call or be interrupted. In some cases it may be interrupted by an external event, e.g. I/O completion, or it may be interrupted by the expiration of its quantum.

(d) [9 marks] Nachos uses a simple round-robin scheduling algorithm. Consider a situation in which n processes are sharing the CPU and the goal is to guarantee each process gets at least one quantum of q ticks out of every T ticks.

Your studies have shown that the average context switching time between processes is s ticks, and the average amount of time an *I/O-bound* process uses before generating an I/O request is t ticks ($t >> s$).

(i) What is the maximum quantum size possible if each process is to receive at least one quantum of CPU time every T ticks?

To service each of n processes we will do n context switches consuming ns ticks and leaving $T - ns$ for useful work. That sets the maximum quantum size at $\frac{(T-ns)}{n}$.

(ii) Briefly describe the impact of each of the following settings for the quantum, q, on achieving the stated goal.

1. $q = s$

2. $q = t$
3. $q > t$

The processes are I/O bound, thus we know that the time on the I/O device is greater than t.

1. $q = s$
   We would expect to spend 50% of our time doing context switches. This is unnecessary overhead.
2. $q = t$
   Hopefully n is small enough that $nq < (T - ns)$, i.e. on average each process gets through the CPU within time T. Normally we would expect $nq << (T - ns)$ with I/O bound processes so this will lead to unnecessary context switches.
3. $q > t$
   This should be the case, especially in an I/O bound system. The additional context switches due to quantum expiration should only be necessary to cut off long bursts.

Question 4. Memory Management [25 marks]

You are part of a design team working on the memory management software for a multitasking operating system that uses paging. You have just been asked to come up with a design for a *type* to keep track of which memory pages are free and which are in use. The type is intended to be used to allocate physical page frames to processes requesting memory.

(a) [4 marks] Identify *two* crucial concerns that you would have in developing this type.

Expect two of:

- *Concurrency control for a shared resource.*
- *Deadlock prevention in resource allocation.*
- *Integrity of the data structure*

(b) [6 marks] What functions would be included in your interface?

constructor and destructor;
allocate N pages (or a way to support the allocation of N pages);
deallocate N pages; and
return the number of free pages...

(c) [7 marks] How was the design of the interface specified in (b) affected by your concerns in (a)? For instance, how would your interface compare with the bitmap provided in Nachos?

The interface would need two major components:

mutual exclusion, i.e. its a monitor, and
batch reservation of pages, for deadlock prevention

This compares with Nachos's page by page allocation which when done by two processes concurrently can lead to deadlock.

(d) [8 marks] Consider a process consisting of seven pages, $\{a, b, c, d, e, f, g\}$. The process has been allocated four frames of physical memory. At time $t = 0$, pages $\{a, b, c, d\}$ are resident. The most recently referenced page was d, the next most recently referenced was a, followed by c and b. The reference string starting at time $t = 1$ and ending at $t = 10$ is:

a e a f b c a f b d

Using the LRU replacement algorithm, and a fixed allocation of four page frames, state for each time, t, from 1 to 10:

1. the resident set, and
2. whether or not a page fault occurred.

The following table shows the resident set (as the stack) and whether or not a page fault occurred.

	a	*e*	*a*	*f*	*b*	*c*	*a*	*f*	*b*	*d*
d	*a*	*e*	*a*	*f*	*b*	*c*	*a*	*f*	*b*	*d*
a	*d*	*a*	*e*	*a*	*f*	*b*	*c*	*a*	*f*	*b*
c	*c*	*d*	*d*	*e*	*a*	*f*	*b*	*c*	*a*	*f*
b	*b*	*c*	*c*	*d*	*e*	*a*	*f*	*b*	*c*	*a*
		pf		*pf*	*pf*	*pf*				

Question 5. File Systems [25 marks]

(a) [4 marks] When a file is opened the kernel places a pointer to the file descriptor in an open file table and returns an integer to the user program.

(i) What is the relationship of the integer to the file descriptor?

The integer is the index into the table where the file descriptor will be found.

(ii) Why does the kernel not return the pointer to the file descriptor to the user program?

A pointer is an address in kernel space and is not valid in user space.

(b) [4 marks] A file system designer has decided to keep the following information in a directory entry:

the file's name,
the file owner's id,
the protection bits,
the size of the file in bytes,
the creation, last modified and last accessed dates, and
a pointer to the file's data on disk.

Discuss this design's suitability. Consider both tree and acyclic graph directory structures.

In a tree each file will have just one directory (parent node). In this case it makes little difference where this information is stored. In a graph multiple directory entries will need to be updated when, for instance, the modified time changes.

(c) For all parts of this question assume a Unix file system with a block size of 4096 bytes.

(i) [4 marks] Show the structure of a file of size 30,000 bytes.

30,000 bytes is 8 blocks. Show an i-node with eight direct pointers.

(ii) [4 marks] Show the structure of a file of size 3,000,000 bytes.

3,000,000 bytes is 733 blocks. That's 12 direct pointers; one indirect pointer to a block with 721 pointers out of a possible 1024.

(iii) [9 marks] Assume a file of size 3,000,000 bytes has been successfully opened.

Explain what happens when the user program reads 36 bytes starting at byte 49,136.

We have to calculate which block(s) the 36 bytes are in. In this case they start at offset 4080 in block 11 and finish in block 12. The operating system would compute the first block number (11) and offset (4080) and then read the data using the direct pointer loaded with inode. Sixteen bytes can be copied to the user memory.

The next block is indirect so the operating system reads the index block and then uses the first entry in that to read the data block. Twenty bytes are copied to user memory.

How many disk operations are required?

All told we did three disk i/os.

Question 6. I/O Subsystems [25 marks]

(a) [6 marks] Modern computer systems have migrated functionality from the operating system kernel to device controllers. Argue the case for and against this change, giving at least two advantages and two disadvantages of placing increased functionality in device controllers.

Advantages: bugs less likely to impact kernel; improved performance; and simplified kernel.
Disadvantages: hard to change (fix bugs, improve); inflexible; must fit kernel view of optimizing performance.

(b) [6 marks] Compare polling with the use of interrupts, stating the advantages and disadvantages of each.

In polling the kernel loops asking the device if it has completed I/O and is ready to transfer data. With interrupts the kernel sets up a handler that is awoken when the device has completed its I/O.

Except for the potential to busy loop polling is much more efficient than interrupts avoiding the transfer of control from an executing program to the kernel. However, interrupts do allow another process to make progress while waiting for an I/O to complete.

(c) [6 marks] In a typical I/O subsystem design, a thread belonging to a user process executes a system call to request an I/O operation and is blocked (waits) in the kernel. The actual I/O operation is done by a kernel thread in the device handler. At the end of the operation, the kernel thread unblocks (signals) the user thread.

What is the advantage of this two threaded structure compared with the user process's thread executing the device handler code?

Using a separate thread allows the device handler code to be written for a single thread removing the need for critical sections in code that is responding to interrupts. For some devices such as disks it also allows optimisation of the device requests.

(d) [7 marks] Assume a disk drive with 3,000 cylinders numbered 0 to 2,999. The drive is currently serving a request at cylinder 143 and the previous request was at cylinder 120. The queue of pending requests in order of arrival is:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position (cylinder 143), give the sequence of requests served and the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms?

(i) SSTF

The SSTF schedule is 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774. The total seek distance is 1745.

(ii) LOOK

The LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86. The total seek distance is 3319.

Question 7. Security [25 marks]

Read the entire question before answering any individual part.

(a) [4 marks] Define the access matrix model for protection.

This model specifies a matrix in which each row is a protection domain (user, time, location) and each column is an object (file, service). The entry in the matrix specifies what access the domain has to the object. The complete matrix specifies the access that each domain has to each object.

(b) [8 marks] Real systems simplify this model by implementing either access lists or capabilities.

(i) Why is simplification seen as necessary?

The entire matrix is both sparse and has a large number of common entries. These approaches attempt to reduce the storage.

(ii) In a few sentences compare *access lists* and *capabilities*.

The access list approach identifies the access a domain has to an object by associating a list of access rights with the object. The capability approach specifies what a domain can access by associated a list of access rights called capabilities with each domain.

(c) [13 marks] Consider a system with the following attributes regarding system security.

- The system has two users, Shaun and Mary.
- The system has one printer, lp.
- The system has two databases, A and B, and a password file, PF.
- Shaun is a systems programmer. Amongst his other activities he is responsible for establishing user accounts. Management has decreed that this activity, and other privileged activities, must take place from his office. Shaun needs to be able to read database A to carry out his job, but has no reason to update either database.
- Mary is in accounts and is responsible for the maintenance of database B and requires data from database A to do this. Mary should not be allowed to print vital corporate information unless she is on site.

(i) Give an access matrix that is consistent with the above requirements and the company policy of not granting access unless it is required.

The following matrix shows the entries of interest.

	lp	*PF*	*A*	*B*
Shaun office		*rw*	*r*	
Mary office	*use*		*r*	*rw*
Mary			*r*	*rw*

(ii) You are given a system with "typical" protection mechanisms, i.e.

user and group IDs, and
file access lists based on owner, group, world.

Could you implement the access matrix you designed for (i)? Give an outline of how you would go about this. If you would not be able to do a complete job indicate where you would have problems and what additional mechanisms might be required to complete a satisfactory implementation.

You need something more than the given facilities to enforce the requirement that Shaun and Mary carry out certain activities from fixed locations. We need the ability to distinguish domains based on more than users and groups and a way of enforcing that.

****