Operating Systems: Problem Sheets

Mike Spivey

Michaelmas Term 1993

The notation [Ta:b] refers to problem no. b in Chapter a of Tanenbaum's book. Text within square brackets in such problems is different from that in the book.

1 Concurrency

1. What is a race condition? [T2:2]
2. Show how counting semaphores (i.e., semaphores that can hold an arbitrarily large value) can be implemented using only binary semaphores [(that can only hold 0 or 1)] and ordinary machine instructions. [T2:6]
3. Round robin schedulers normally maintain a list of all runnable processes, with each process appearing exactly once in the list. What would happen if a process occurred twice in the list? Can you think of any reason for allowing this? [T2:16]
4. Measurements of a certain system have shown that the average process runs for a time T before blocking for I/O. A process switch requires time S, which is effectively wasted (overhead). For round robin scheduling with quantum Q, give a formula for the CPU efficiency for each of the following[, assuming that there is no variation among the processes].
   (a) $Q = \infty$
   (b) $Q > T$
   (c) $S < Q < T$
   (d) $Q = S$
   (e) Q nearly 0. [T2:17]
5. The aging algorithm with [$\alpha = 1/2$] is being used to predict run times. The previous four runs, from oldest to most recent are 40, 20, 40, and 15 msec. What is the prediction of the next time? [T2:21]

6. (For probability experts) The aging algorithm is being used to estimate the mean value of a sequence of observations X_n. Suppose that the distribution of the X_n's does not actually vary over time, so that they are independent random variables, identically distributed with mean μ and variance σ^2. Let Y_n be the n'th estimate computed by the aging algorithm, so that $Y_n = \alpha X_n + (1 - \alpha) Y_{n-1}$. For large n, show that $E(Y_n) \approx \mu$ and find the variance $V(Y_n)$.

7. A batch-processing system needs a module to manage allocation of N line-printers, identified by integers in the range 0 up to $N - 1$. It must provide functions *alloc_printer* and *free_printer* with the follwing headings:

```
/* alloc_printer -- allocate a printer */
PUBLIC int alloc_printer();

/* free_printer -- free printer K */
PUBLIC void free_printer(k)
int k;
```

   Alloc_printer allocates a printer and returns its number; this printer is not allocated to any other client until the same number is passed as an argument to *free_printer*.

   The module must allow safe use by several concurrent client processes, but may assume that these processes honestly call *free_printer* for printers that they have previously obtained by calling *alloc_printer*. It may ignore problems of deadlock.

   Implement the allocation module

   (a) Using a server process and MINIX-style messages,

   (b) Using a monitor and condition variables,

   (c) Using semaphores.

   [Hints: your module will need to keep track of the set of free printers, perhaps representing it as an array of Booleans. You may also find it convenient to keep track of the size of this set. Some solutions will need to maintain an explicit queue of processes that are waiting for a printer to become free.]

8. In the lecture, it was shown how to implement message-passing using semaphores. Show how a semaphore can be implemented as a process that communicates with MINIX-style messages. Harder: show how to

use a *pair* of MINIX processes to implement a semaphore without an explicit queue of processes waiting to complete a *down* operation. [This doesn't give a practical way of implementing semaphores, but it shows that semaphores and messages are equally expressive.]

2 Input/Output

1. A disk is double interleved, [so that each sector n of a track is separated from sector $n + 1$ by two other sectors]. It has eight sectors of 512 bytes per track, and a rotation rate of 300 rpm. How long does it take to read all the sectors of a track in order, assuming the arm is already positioned, and 1/2 rotation is needed to get sector 0 under the head? What is the data rate? Now repeat the problem for a [non-interleaved] disk with the same characteristics. How much does the data rate degrade [owing] to interleaving? [T3:3]

2. In which of the [three] I/O software layers is each of the following done[?]

   (a) Computing the track, sector, and head for a disk read.

   (b) Maintaining a cache of recently used blocks.

   (c) Writing commands to the device registers.

   (d) Checking to see if the user is permitted to use the device.

   (e) Converting binary integers to ASCII for printing.

3. A salesman claimed that his company's version of UNIX] used the elevator algorithm, and also queued multiple requests within a cylinder in sector order [for speed]. [A student] wrote a program to randomly read 10,000 blocks spread across the disk. [The performance was identical to that] expected from first-come first-served. Was the salesman lying? [T3:21]

4. Consider how a terminal works. The driver outputs one character and then blocks. When the character has been printed, an interrupt occurs and a message is sent to the blocked driver, which outputs the next character and then blocks again. If the time to pass a message, output a character, and block is 4 msec, does this method work well on 110 baud lines? How about 4800 baud lines? [T3:26]

5. Stripped of all error handling and most of the detail, the MINIX floppy disk driver is a process like this:

```
PUBLIC void floppy_task()
{
     int caller, command;
     int cylinder, sector;
     message m;

     while (TRUE) {
          receive(ANY, &m);
          caller = m.m_source;

          switch (m.m_type) {
          case READ:  command = DISK_READ;  break;
          case WRITE: command = DISK_WRITE; break;
          default:
               panic("Floppy disk got bad message");
          }

          cylinder = m.m_block / SECS_PER_CYL;
          sector = m.m_block % SECS_PER_CYL;
          floppy_command(SEEK, cylinder);
          receive(HARDWARE, &m);
          floppy_command(command, sector);
          receive(HARDWARE, &m);

          m.m_type = REPLY;
          send(caller, &m);
     }
}
```

This code uses calls to *receive* with argument *HARDWARE* to wait for the interrupt that says the disk command is finished, without accepting prematurely another read or write request. What changes would be needed

(a) if this selective *receive* were not available?

(b) if the floppy disk driver were implemented not as a separate process, but as a monitor with two entry procedures, *floppy_request* called to read or write a block, and *floppy_int* called when a floppy disk interrupt occurs?

3 Memory management

1. Using the model of Fig. 4.2, [that at any time, the events that different processes are waiting for I/O are independent,] we can predict the increase in throughput as a function of the degree of multiprogramming. Suppose that a computer has a 2M memory, of which the operating system takes 512K [...] and each user program also takes 512K. If all programs have 60 percent I/O wait, by what percentage will throughput increase if another 1M is added? [T4:2]

2. In a swapping system with variable partitions, the segment [sizes are independent and exponentially distributed with probability density function] $e^{-s/10}/10$, where s is the segment size in kilobytes. The holes sizes have probability density $e^{-h/5}/5$, where h is the hole size in kilobytes. [When the system is in statistical equilibrium, w]hat is the average fraction of wasted memory? [T4:5]

3. A computer whose processes have 1024 pages in their address space keeps its page tables in memory. The overhead required for reading a word from the page table is 500 nsec. To reduce this overhead, the computer has an associative memory, which holds 32 (virtual page, physical page frame) pairs, and can do a look up in 100 nsec. What hit rate is needed to reduce the mean overhead to 200 nsec? [T4:11]

4. [For a particular program,] it has been observed that the number of instructions executed between page faults is directly proportional to the number of page frames allocated to [the] program. [...] Suppose that a normal instruction takes 1 microsec, but if a page fault occurs, it takes 2001 microsec. If [the] program takes 60 sec to run, during which time it gets 15,000 page faults, how long would it take to run if twice as much memory were available? [T4:18]

5. In the lecture, we proved that at statistical equilibrium, the number of holes in memory is half the number of processes, but ignored the possibility that an allocation would consume all of a hole. Suppose now that the memory allocator is modified so that an allocation that almost consumes a hole is rounded up to take all of it – this has the effect of reducing the number of tiny holes and speeding up the allocator. Suppose that the probability that a complete hole is taken in an allocation is p. Compute the ratio of holes to processes under these modified conditions.

6. In implementing 'best fit' allocation, what data structures and algorithms could be used to avoid having to search the entire hole list on

every allocation?

7. In the [book's] implementation of MINIX, when [a new program is loaded], the memory manager checks to see if a hole large enough to contain the new memory image is available. If not, the call is rejected. A better algorithm would be to see if a sufficiently large hole would be available after the current memory image was released.

   [This change could be accomplished by replacing line 6135:

   ```
   if (text_clicks + tot_clicks > max_hole())
        return EAGAIN;
   ```

   with the new line

   ```
   if (text_clicks + tot_clicks >
             future_hole(old_base, old_clicks))
        return EAGAIN;
   ```

   (for suitable values of *old_base* and *old_clicks*), and writing a function *future_hole(b, s)* that returns the size of the largest hole that will exist after the block with base b and size s is released.

   By studying the memory allocation code on pages 573–7 (including *max_hole* at line 7131), implement this function.]

4 File systems

1. Free disk space can be kept track of using a free list or a bit map. Disk addresses require D bits. For a disk with B blocks, F of which are free, state the condition under which the free list uses less space than the bit map. For D having the value 16 [...], express your answer [in terms of the] percentage of the disk space that [is] free. [T5:3]

2. The link count in the UNIX i–nodes is redundant. All it does is tell how many directory entries point to the i–node, something that could equally well be seen by looking at the directories. Why is this field used? [T5:6]

3. A file system checker has built up its [bit-maps for file system blocks]. They are:

   ```
   In use:  1 0 1 0 0 1 0 1 1 0 1 0 0 1 0
   Free:    0 0 0 1 1 1 0 0 0 1 0 1 1 0 1
   ```

Are there any errors? If so, are they serious? Why? [T5:9]

4. It has been suggested that the first part of each UNIX file be kept in the same disk block as its i–node. Would this be a good thing to do? [T5:10]

5. Three different protection mechanisms that we have discussed are capabilities, access control losts, and the UNIX *rwx* bits. For each of the following protection problems, tell which of these mechanisms can be used.

   (a) Rick wants his files readable by everyone except Jennifer.

   (b) Helen and Anna want to share some secret files.

   (c) Cathy wants some of her files to be public.

6. Can a Trojan Horse attack succeed in a system protected by capabilities? [T5:24]

7. Suppose a file's protection bits are `------r--` (so the owner has no permissions, but 'others' have read permission). By studying the MINIX source code, or by experimenting with SunOS, find out if the owner read the file. How could the MINIX code be modified so that the opposite convention applies?

8. How could MINIX be modified to have 'exclusive' files that can only be opened by one process at a time?

5 Formal specification

A module manages allocation of printers to operating system processes. The module keeps track of which process is using which printer; its state is a partial function from printers to the process id's of the current users:

Manager

$$user : PRINTER \pfun PID$$

The operation of allocating a printer is specified as follows:

Allocate

$Manager; Manager'$
$caller? : PID$
$pr! : PRINTER$

$\mathrm{dom}\, user \neq PRINTER$

$pr! \in PRINTER \setminus \mathrm{dom}\, user$
$user' = user \oplus \{pr! \mapsto caller?\}$

Question 1 *Explain the significance of the three predicates in the body of this schema.*

The *Release* operation for freeing a printer is specified like this:

Release

$Manager; Manager'$
$caller? : PID$
$pr? : PRINTER$

$\ldots$

Question 2 *Determine informally the pre-condition for successful completion of a Release operation and the post-condition that describes how the state changes.*

Question 3 *Express these predicates in mathematical notation (mine or your own).*

Question 4 *Specify a ReleaseAll operation that releases all printers held by the calling process.*

The installation manager now decides that no process may use more than one printer at a time.

Question 5 *Express this condition as an invariant on the state of the module*

Question 6 *Calculate the extra pre-condition is placed on the Allocate operation.*