Operating Systems 3SH3 Homework 1

Bradley Kohler

Question 1.7

In virtually all systems that include DMA modules, DMA access to main memory is given higher priority than processor access to main memory. Why?

Answer 1.7

The DMA module must take control of the bus when reading from or writing to main memory. This is because the data does not go through the processor but instead bypasses. While the DMA takes control of the bus the processor must wait for the DMA module. This means that the DMA module has higher priority than the processor. The DMA will cause the processor to run more slowly when accessing main memory. Regardless, this is one of the quicker ways of accessing the main memory.

Question 1.10

Consider the following code:

```
for (i = 0; i < 20; i++)
        for (j = 0; j < 10; j++)
                a[i] = a[i] * j
```

a) Give one example of the spatial locality in the code.
b) Give one example of the temporal locality in the code.

Answer 1.10

a) An example of spatial locality is the instructions. Every instruction in the code is followed by another instruction. These are accessed sequentially.
b) An example of temporal locality is the loop. The processor is executing the same set of instruction repetitively.

Question 1.12

Consider a memory system with the following parameters:

Tc = 100 ns Cc = 0.01 cents/bit

Tm = 1,200 ns Cm = 0.001 cents/bit

a) What is the cost of 1 MByte of main memory?
b) What is the cost of 1 MByte of main memory using cache memory technology?
c) If the effective access time is 10% greater than the cache access time, what is the hit ratio H?

Answer 1.12

a) $1\text{MByte} \times 0.001\frac{\text{cents}}{\text{bit}} = 8000000\text{bits} \times 0.001\frac{\text{cents}}{\text{bit}} = 80\$$
b) $1\text{MByte} \times 0.01\frac{\text{cents}}{\text{bit}} = 8000000\text{bits} \times 0.01\frac{\text{cents}}{\text{bit}} = 800\$$

c) $\frac{T_c}{T_s} = \frac{1}{1+(1-H)\times\frac{T_m}{T_c}}$

$$T_s = 1.1 \cdot T_c$$
$$1.1 \cdot T_c = T_c + (1-H)T_m$$
$$1.1 \cdot (100) = 100 + (1-H)(1200)$$
$$1.1 = (13 - 12H)$$
$$\frac{1190}{1200} = H$$
$$H = 0.992$$

Question 1.13

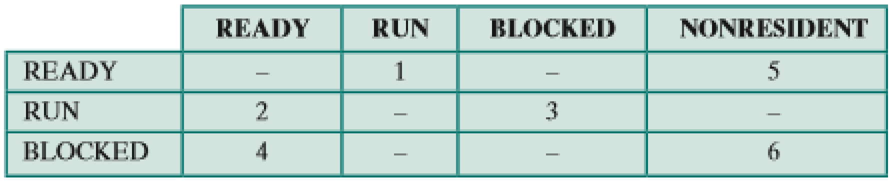
A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20 ns are required to access it. If it is in main memory but not in the cache, 60 ns are needed to load it into the cache (this includes the time to originally check the cache), and then the reference is started again. If the word is not in main memory, 12 ms are required to fetch the word from disk, followed by 60 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.9 and the main-memory hit ratio is 0.6. What is the average time in ns required to access a referenced word on this system?

Answer 1.13

$$Avg = (0.9)(20ns) + (0.1)(0.6)(80ns) + (0.1)(0.4)(12000080ns) = 480026ns$$

Question 3.1

The following state transition table is a simplified model of process management, with the labels representing transitions between states of READY, RUN, BLOCKED, and NONRESIDENT.



	READY	RUN	BLOCKED	NONRESIDENT
READY	–	1	–	5
RUN	2	–	3	–
BLOCKED	4	–	–	6

Give an example of an event that can cause each of the above transitions. Draw a diagram if that helps.

Answer 3.1

1. Ready to Run occurs once the program has been scheduled. The program is allocated to the CPU.
2. Run to Ready can occur if a program's time slice expires. The program can be interrupted and moved back to the ready state.
3. Run to Blocked can occur if an OS process requests it or I/O or kernel.
4. Blocked to Ready can occur if the held event completes an event it was waiting for.

5. Ready to Non-Resident can occur if memory is overloaded which results in a process being temporarily swapped out of memory.
6. Blocked to Non-Resident can occur for the same reasons as number 5.

<u>Question 3.2</u>

Assume that at time 5 no system resources are being used except for the processor and memory. Now consider the following events:

At time 5: P1 executes a command to read from disk unit 3.

At time 15: P5's time slice expires.

At time 18: P7 executes a command to write to disk unit 3.

At time 20: P3 executes a command to read from disk unit 2.

At time 24: P5 executes a command to write to disk unit 3.

At time 28: P5 is swapped out.

At time 33: An interrupt occurs from disk unit 2: P3's read is complete.

At time 36: An interrupt occurs from disk unit 3: P1's read is complete.

At time 38: P8 terminates.

At time 40: An interrupt occurs from disk unit 3: P5's write is complete.

At time 44: P5 is swapped back in.

At time 48: An interrupt occurs from disk unit 3: P7's write is complete.

For each time 22, 37, and 47, identify which state each process is in. If a process is blocked, further identify the event on which it is blocked.

<u>Answer 3.2</u>

<u>Time 22</u>

- P1 is in the blocked state. P3 puts P1 in blocked state by requesting a read from same disk unit.
- P3 is in the run state.
- P5 is in the ready state.
- P7 is in the blocked state. P1 puts P7 in blocked state by reading from the same disk unit.
- P8 is in the run state.

<u>Time 37</u>

- P1 is in the ready state.
- P3 is in the ready state.
- P5 is in the blocked state. P7 puts P5 in blocked state by writing to same disk unit.

- P7 is in the blocked state. Unable to execute until write is finished in disk unit.
- P8 is in the run state.

<u>Time 47</u>

- P1 is in the run state.
- P3 is in the run state.
- P5 is in the ready state.
- P7 is in the run state.
- P8 is a zombie.

<u>Question 3.3</u>

Figure 3.9b contains seven states. In principle, one could draw a transition between any two states, for a total of 42 different transitions.

a) List all of the possible transitions and give an example of what could cause each transition.
b) List all of the impossible transitions and explain why.

<u>Answer 3.3</u>

a)

New to Ready: A process is newly created and stored in main memory.

New to Ready/Suspend: A process is newly created and stored in secondary memory.

Ready to Ready/Suspend: A process is in main memory and moved to secondary memory.

Ready/Suspend to Ready: A process is in secondary memory and moved to main memory.

Blocked/Suspend to Ready/Suspend: A process is in secondary memory and an event occurs that it had been waiting for.

Blocked to Blocked/Suspend: A process has been blocked by another process/ I/O / kernel and has been moved to secondary memory since it is waiting for an event.

Blocked/Suspend to Blocked: An event occurred that the process was waiting for. The process is moved to main memory. However, the process is still blocked by another process / I/O / kernel.

Blocked to Ready: An event occurs that deems the process to be unblocked.

Ready to Running: The scheduler schedules the process to the CPU.

Running to Ready: The process’ time slice expires.

Running to Ready/Suspend: The process’ time slice expires and is stored in secondary memory until space is available in main memory.

Running to Blocked: The process must wait for another process / I/O / kernel.

Running to Exit: The program finishes executing.

b)

Not Running to Exit: Any process must go through the CPU to exit.

Not Ready to Running: Any process must be ready to move to a running state. This is because the process must be waiting in main memory for the scheduler or dispatcher.

Exit to Any State: A process cannot finish and then move back into the OS. Only new process' move into the OS.

Any State to New: A created process cannot be new.

Running to Blocked/Suspend: An alternate process must decide if a blocked process should be suspended.

Blocked to Running: The dispatcher only looks at processes in the ready state.

Blocked to Ready/Suspend: This must be done in 2 stages. We cannot simultaneously suspend and block a process.

Blocked/Suspend to Ready: Same reason as Blocked to Ready/Suspend.

Blocked/Suspend to Running: Same reason as Blocked to Running.

<u>Question 3.4</u>

For the seven-state process model of Figure 3.9b, draw a queueing diagram similar to that of Figure 3.8b.

<u>Answer 3.4</u>

<u>Question 3.5</u>

Consider the state transition diagram of Figure 3.9b. Suppose that it is time for the OS to dispatch a process and that there are processes in both the Ready state and the Ready/Suspend state, and that at least one process in the Ready/Suspend state has higher scheduling priority than any of the processes in the Ready state. Two extreme policies are as follows: (1) Always dispatch from a process in the Ready state, to minimize swapping, and (2) always give preference to the highest-priority process, even though that may mean swapping when swapping is not necessary. Suggest an intermediate policy that tries to balance the concerns of priority and performance.

Answer 3.5

Give a penalty to the processes in the ready/suspend state. This is so that the processes in the ready/suspend state are only in priority if they have a much greater priority than the processes in the ready state.

Question 4.4

Consider an environment in which there is a one-to-one mapping between user-level threads and kernel-level threads that allows one or more threads within a process to issue blocking system calls while other threads continue to run. Explain why this model can make multithreaded programs run faster than their single-threaded counterparts on a uniprocessor computer.

Answer 4.4

In a multithreaded program one KLT can make a blocking system call while the other threads continue to run. On uniprocessors, a single-threaded process would have to block all of its calls where multithreaded process threads can continue.

Question 4.7

Many current language specifications, such as for C and C++, are inadequate for multithreaded programs. This can have an impact on compilers and the correctness of code, as this problem illustrates. Consider the following declarations and function definition:

```
int global_positives = 0;

typedef struct list {  struct list *next;  double val; } * list;

void count_positives(list l) {

        list p;

        for (p = l; p; p = p -> next)

                if (p -> val > 0.0)

                        ++global_positives;

        }
```

Now consider the case in which thread A performs

count_positives(<list containing only negative values>);

while thread B performs

++global_positives;

a) What does the function do?

b) The C language only addresses single-threaded execution. Does the use of two parallel threads create any problems or potential problems?

<u>Answer 4.7</u>

a) The function counts the number of positive values in a list.
b) This should work. Count_positives does not update global_positives and so the two threads operate on global data and do not require locking.

<u>Question 4.8</u>

But some existing optimizing compilers (including gcc, which tends to be relatively conservative) will "optimize" count_positives to something similar to

```
void count_positives(list l)

{

        list p;

        register int r;

        r = global_positives;

        for (p = l; p; p = p -> next)

                if (p -> val > 0.0) ++r;

                        global_positives = r;

}
```

What problem or potential problem occurs with this compiled version of the program if threads A and B are executed concurrently?

<u>Answer 4.8</u>

The update of global_positives by thread B may be lost because thread A may write back an earlier value of global_positives.

<u>Question 4.9</u>

a) What does this program accomplish?

b) Here is the output from the executed program:

```
$ ./thread2 ..o.o.o.o.oo.o.o.o.o.o.o.o.o.o.o..o.o.o.o.o

myglobal equals 21
```

Is this the output you would expect? If not, what has gone wrong?

Answer 4.9

a) This program creates a new thread. The main thread and the newly created thread increment the global variable myglobal 20 times.
b) No. Each thread should increment by 20. So myglobal should be 40. This happens because in the new thread myglobal is copied to j then incremented and then j is copied to myglobal. In the main thread, if myglobal is incremented and then j is copied to my global then the value was only incremented once in each thread and main does an overwrite on the modification the new thread made.