HW - 2.2 - HW 23

Questions: Tannenbaum & Bos 2.{10,14,17,25,43,44,45,46}

**10.** In the text it was stated that the model of Fig. 2-11(a) was not suited to a file server

using a cache in memory. Why not? Could each process have its own cache?

In the figure 2-11(a) we have traditional processes with their own address space and single thread. Fig. 2-11(b) shows multithreading which is used for threads in the same process working together on the same job. A cache would be better applicable to similar tasks. Single thread performance is dependent on the application and the set of data. Threads can share a cache when in multithreading, which is better cache utilization. A cache is used by a CPU or multiple CPUs share a cache.

**14.** In Fig. 2-12 the register set is listed as a per-thread rather than a per-process item.

Why? After all, the machine has only one set of registers.

Each thread contains its own stack. Each thread calls different procedures and each thread has a different execution history. Threads share address space within a process and one thread can change the contents of another threads stack. Per-thread items are for when threads share a job and are closely cooperating with each other. Per-process items are used when we have processes that are unrelated.

**17.** In this problem you are to compare reading a file using a single-threaded file server

and a multithreaded server. It takes 12 msec to get a request for work, dispatch it, and

do the rest of the necessary processing, assuming that the data needed are in the block

cache. If a disk operation is needed, as is the case one-third of the time, an additional

75 msec is required, during which time the thread sleeps. How many requests/sec can

the server handle if it is single threaded? If it is multithreaded?

If single threaded file server is used, cache misses take 87 msec. while hits take 12 msec. (12\*2/3) + (87\*1/3) = 37 weighted average. 1000 msec. / 37 msec. = 27.027027 requests/sec. Multithreaded does not wait so 1000 msec. / 12 msec. = 83 1/3 requests/sec.

**25.** Can the priority inversion problem discussed in Sec. 2.3.4 happen with user-level

threads? Why or why not?

No, although possible in kernel threads, user-level threads cannot be preempted unexpectedly by a process of lower priority.

**43.** Measurements of a certain system have shown that the average process runs for a time

*T* before blocking on I/O. A process switch requires a 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:

(a) *Q* = ∞

(b) *Q* > *T*

(c) *S* < *Q* < *T*

(d) *Q* = *S*

(e) *Q* nearly 0

CPU efficiency = CPU time/CPU total time.

1. T/(S+T) If Q>=T, the process runs for T time and a process switch S.
2. T/(S+T)
3. T/(T +(S\*T/Q) = Q/(Q+S) -> (time/ (time + wasted time) )
4. T/(T +(Q\*T/Q) = ½
5. As Q approaches 0, efficacy goes to zero.

**44.** Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and *X*. In what

order should they be run to minimize average response time? (Your answer will

depend on *X*.)

Completing the shortest job first will minimize average response time X. The time elapsed between end of inquiry and beginning of response time.

0<X<=3 = X,3,5,6,9

X<=5 = 3,X,5,6,9

X<=6 = 3,5,X,6,9

X<=9 = 3,5,6,X,9

9<X = 3,5,6,9,X

**45.** Five batch jobs. *A* through *E*, arrive at a computer center at almost the same time.

They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined)

priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority.

For each of the following scheduling algorithms, determine the mean process

turnaround time. Ignore process switching overhead.

(a) Round robin.

(b) Priority scheduling.

(c) First-come, first-served (run in order 10, 6, 2, 4, 8).

(d) Shortest job first.

For (a), assume that the system is multiprogrammed, and that each job gets its fair

share of the CPU. For (b) through (d), assume that only one job at a time runs, until it

finishes. All jobs are completely CPU bound.

1. Round robin (31 + 28 + 23 + 17 + 8)\* (1/5) = 21.4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| A | B | C | D | E | A | B | C\* | D | E | A | B | D | E | A |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| B | D\* | E | A | B | E | A | B\* | E | A | E | A | E\* | A | A | A\* |

1. Priority starting at B = 5, (6 + 14 + 25 + 27 + 31)/5 = 20.6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1-6 | 7-14 | 15-25 | 26-27 | 28-31 |
| B | E | A | C | D |

1. FCFS (11 + 17 + 19 + 23 + 31) /5 = 20.2 or = (5\*11 + 4\*6 + 3\*2 + 2\*4 + 1\*8)/5 THE OTHER WAY Summation of multiplying times and priorities, divided by amount.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1-11 | 12-17 | 18-19 | 20-23 | 24-31 |
| A | B | C | D | E |

1. SJF (2 + 6 + 12 + 20 + 31)/5 = 14.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1-2 | 3-6 | 7-12 | 13-20 | 21-31 |
| C | D | B | E | A |

**46.** A process running on CTSS needs 30 quanta to complete. How many times must it be

swapped in, including the very first time (before it has run at all)?

quantum 1, 2, 4, 8, 15 ; 5 times total swapped in. Multi-level scheduling algorithm assigns each user program as it enters the system. If not completed, it is put on the end of the next level queue and so on reading from the queue.