Smith:

1. What advantages (if any) are there in having different quantum sizes on different levels of a multilevel foreground-background scheduling algorithm? Describe how you would use different quantum sizes to optimize performance. (10)

>Multilevel foreground-background systems consist of a series of queues. Jobs in the highest priority (foreground) queue(s) have priority over those in lower priority queues. Frequently, jobs are first run in the highest priority queue, and if they exceed their quantum, are demoted to successively lower queues. Jobs that do not exceed their quantum (e.g. I/O intensive jobs) are typically returned to the highest priority queue after returning from being blocked. Frequently, the queue length in the highest priority queue is short, so that high priority jobs can get immediate service, and if short, can complete quickly. Queue lengths for lower priority queues are typically longer, so that (a) the overhead of starting the job (e.g. moving it into the in-memory queue) can be amortized over a longer run interval; (b) jobs requiring longer run times will receive larger amounts of service (due to the longer queue length) and so will make progress toward completion.

Smith:

2. What's the difference between internal and external fragmentation? Which applies to segmentation (without paging) and which applies to a pure paging system? Explain. Also explain the possible effect of each on system performance. (10)

>Internal fragmentation occurs when a block is allocated that is larger than the data it must contain. This applies to paging, since pages are fixed size. The range of virtual addresses used by a process may not be contiguous, and each range of used addresses will typically occupy only part of one or more blocks. Even if a contiguous region of virtual space is used, part of the last page will be unused.

External fragmentation occurs when variable size blocks are allocated and deallocated in a free storage area. Small regions of unallocated space will occur between allocated regions. The small unallocated spaces will be too small to satisfy storage requests, and thus will be wasted. This occurs with segmentation, since segments are of variable size.

Either type of fragmentation will cause wasted memory space, which will on the average decrease the average level of multiprogramming, and therefore lead to more multiprogramming idle. External fragmentation will also occur overhead for allocation using free lists, and possibly for compaction (which can be quite slow) if it is used. Another performance loss may occur because, since there is less useful main memory available, more transfers of pages or segments between main memory and disk will need to occur.

3. Assume that the size of your address space (in pages) is X, that the amount of address space (in pages) used by a process is Y, and that the size of the physical memory of the machine (in page frames) is Z. Assume X, Y and Z are known and fixed; i.e. they don't change as the process executes. Assume that you have a hashed page table of size (number of entries) A
for each process that is used to map pages of the process to page frames. Assume that you also have a second hashed page table of size (number of entries) B for each
process that is used to map pages that are not in memory to disk block addresses. Each of A and B should be (at least) proportional to which of the following (without being larger than necessary or useful); explain your answer for each of A and B: (12)

X, Y, Z, min(X,Y), max(X,Y), min(X,Z), max(X,Z), min(Y,Z), max(Y,Z), min(X,Y,Z), max(X,Y,Z)

From cs162-tb@pasteur.EECS.Berkeley.EDU Wed Oct 11 14:55:32 2000
From: "Z. Morley Mao" <cs162-tb@pasteur.EECS.Berkeley.EDU>

X is the size of the entire address space in pages. Y is the size of virtual memory address space in pages for each process. Z is the size of physical memory of the machine in page frames. Hashed page table A maps a virtual address to a physical address. Therefore, A should be at least proportional to min(Y,Z). Hashed page table B maps virtual address to disk block address. The disk block address must contain all the virtual memory address space, because everything can be paged out to disk. Therefore, B should be at least proportional to Y.

4. In order to link and load object code modules, you need several tables. List all of the tables that you use (and explain what information is

contained in each, including the fields for each entry). Describe the process by which several object code modules are linked and loaded into memory. (22)

From ushankar@cs.berkeley.edu Thu Oct 12 16:02:34 2000

This was basically the information from the lecture notes on linking/loading.

Tables:
1. Segment Table - [Segment Name, Segment Size, Nominal base]

2. Symbol table - contains global definitions - has table of labels that are needed in other segments. Typically global variables, functions.
[symbol, segment_name, offset from base of segment]

3. Relocation Table - table of addresses within this segment that need to be fixed, i.e. relocated. Contains internals - references to locations within this segment, and external references - references that are believed to be external. (I.e. we didn't find them here.) [address location, symbol name, offset to symbol (i.e. offset to address), length of address field]

Process of linking/loading:
1) Determine location of each segment
2) Calculate values of symbols and update symbol table
3) Scan relocation table and relocate addresses.

Obviously, some more detail was needed in the process part. I gave full credit for reasonably complete, concise, and well-explained answers: things that were nice to have were protection bits for segments, possibly merging segments together, changing base address at load-time (and maybe having to do another round of address fix-up).

Points were deducted for missing a table, for incorrectly stating what a table was for, for bringing page tables and virtual memory into it, and for missing or adding parts to the process of linking/loading.

No additional points for dynamic linking.

5a. You have the following jobs arriving at the indicated time, and requiring the indicated amount of processing. For each of FIFO, SJF and SRPT, show on a time line which job is running at each instant. Compute the mean flow time for each algorithm. Show your work. (12)

job	arrival time	service time
1	0	10
2	7	5
3	9	7
4	10	21
5	25	2

From mukunds@EECS.Berkeley.EDU Wed Oct 11 16:38:17 2000

FIFO : mean flow time = 16.8

Job executing	Time Span
1	0-10
2	10-15
3	15-22
4	22-43
5	43-45

SJF (This is Non-preemptive) - Everything identical to FIFO

SRPT (This is Preemptive) - Mean Flow Time = 13.6

Job executing	Time Span
1	0-10
2	10-15
3	15-22
4	22-25
5	25-27
4	27-45

5b. Assume the same set of jobs as in 5a. Assume now that you have two processors, and that your objective is now to minimize the time until all of the jobs are completed. Jobs may not be preempted. Please show a time line (actually 2 time lines, one for each processor) with the (an) optimal schedule. Any schedule consistent with the above assumptions and the

data given is acceptable. (10)

Umesh:

Full credit for any solution which 1) ended at t=31, 2) started each process at or after its arrival time, 3) only ran one process at a time on a given processor, and 4) did not run the same process at the same time on both. 6 points for solutions completing at t=33. Fewer points for slower solutions, or for those which violated any of the above constraints.

6. For the following two cases, please either show a complete safe sequence or prove that there isn't one (12):

Process	has-X	has-Y	max-needs-X	max-needs-Y
A	10	20	75	50
B	0	70	50	90
C	30	10	60	40
D	50	80	100	220

(a) available: X: 40 Y: 40
(b) available: X: 40 Y: 35

(a) CBAD or CABD.

This table shows what more of X and Y each process needs in order to make progress. This value is obtained by subtracting has-X(Y) from max-needs-X(Y).

Process	need-more-X	need-more-Y
A	65	30
B	50	20
C	30	30
D	50	140

Now we can let a process complete if its need-more-X <= available-X and need-more-Y <= available-Y.

This table shows the available X and available Y after a give process finishes.

Process:	available(X)	available(Y)
C	70	50
A	80	70
B	80	140
D		

the above table shows that CABD is a safe sequence. Similarly CBAD is also safe.

(b)
There is no safe sequence, since D needs 220 Y in order to make progress, but there is only 215 Y available.

7. Why is minimizing the variance of the flow time (var(f(i)) not the same as minimizing the variance of (f(i)/s(i))? Which scheduling algorithm does the former? Which is better to minimize for scheduling processes in computer systems? Why? (Note that (f(i) is the flow time of job i; s(i) is the service time of job i.) (12)

From mukunds@EECS.Berkeley.EDU Wed Oct 11 16:38:17 2000

Minimising var f(i) is not the same as min var f(i)/s(i), because job service times are different and the former does not depend on job service times, while the later tries to finish shorter jobs sooner and longer jobs later.

FIFO does the former.

In computer systems, the latter is better to minimise, because this ties in with user's expectations and notions of predictability, where they expect short jobs, like "ls" to finish quickly, but are willing to wait longer for long jobs like some computation-intensive number crunching task.