þÔþœp□¦4Ýä÷¬©
l|ŠÕRu□□<ö6Œ,-□□Ì0:ÀhÇ/>*c>Q*8¿âB□5|,IçÜüwù□□ÏÝŸ9□õ¤=Ì-u|²¬¬|□□¯J"□²¿çŽžò*8□1/Ú±|\Î□T¥\R□Ëæi:·Ç□"8uh□□ZÆßO¶eç†Á
æV?_ãÆ"vD±¿□(YÜ˜□□□è%èò(ô□[iÝ,Ûj
®žê'A¿□'f□)?Nœ'□9□ÖB□uf1Ž)0Ç\""€□□1M7º-□z□T□e□ó2ý6'aN;JÓ]èi»ýÀPÛß^x‡Ùs¥(&Ã□.(Í□Ú□Þ□t3o/□|□Ì□;¬W•□ÙS%ø>ØÕŒ□□ÕÝÊÀ
uc"øHG}w-m)B78□%7□¢©Æ6Î×kw²I□ÖM-6|8Ž:*%Ë¹àØ)°□I"□NáÂæX/øè□á¬É□O□æ»þ%¤šæp„Îç-=Òí@û□Qsv□ô§ãÞe(iÎM□*Få□□|ƒ□Ät;-üý
pÖ
¬Ýï|□³>5ÃËÍŽ

AAYUSH GUPTA

axg354

77

Mid-term Exam: Operating Systems, CSE 511

Nov. 8, 2007 (duration: 2 hours)

General instructions:

- This exam has FIVE questions. Make sure your exam book has all of these.
- Do not forget to put down your name on the exam book.
- Explain your answers clearly and be concise. *Do not write long essays.*
- Be a smart test-taker. If you get stuck, move on to the next question.
- State any assumptions you make clearly.
- Good luck!

---

1. **Processes and related abstractions** (5+5+5 = 15 pts)

   (a) In Linux, there are two separate stacks for each process - a *user mode* stack and a *kernel mode* stack. Why are there two stacks?

   (b) In Linux, the kernel mode stack is maintained within the same `struct` that holds the PCB. What benefit(s) does this offer?

   (c) Linux reserves only 8KB for the kernel mode stack. What do you think happens when this stack overflows?

In general, for any kernel functions

Kernel mode stack stores information regarding the state of execution of a thread when it goes from user-mode to kernel mode. It also contains scheduling & accounting information used by the kernel. Whenever an interrupt occurs the thread switches from user-mode to kernel mode & the kernel mode stack is used to set up user-mode stack for signal handler & perform signal handling functions like setup-frame ( ) etc.

2/5

The question was WHY not what these stacks do.

É9Ÿ%È1òoÒ%Ò/²±YdO÷ÞÊÅ5ä-Uk®ÜÍŠ%¯1;JÂ€[(f6SZ
[PÍ'%ZZ<]°ÕŸ÷)F.9h.«ci·ƒ-¤8

User mode stack contains all the user
defined variables whereas when the
thread runs in kernel-mode it uses the

thread's kernel mode stack & helps
Kernel-mode processing
in ~~faster~~

b)

- Easier management as it is easier to attribute a kernel mode stack of a process if its in the same place as its PCB

OK

- The kernel can easily locate the kernel mode stack of a process in O(1) time & also prevents duplication of features (reduces redundancy)

Why would there be a duplication otherwise?

4/5

More benefits — See solutions

ћ¤
□□ÌÀ'9#zy¹gYcF□æ8^W¯å^ÛÚTõ□

(c) Each thread is allocated a kernel mode stack so increasing the size of the stack by even a small amount will drastically increase the memory footprint of the system.

Linux might crash if the kernel stack overflows. This might happen if recursive calls are stored in kernel mode stack.

5/5

$\frac{S_A}{5}$

1

2 $5/2$ $5/2$

2. **CPU scheduling** (10+5=15 pts)

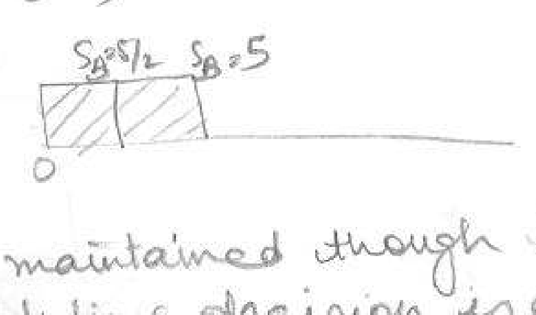
(a) Consider a computer in which the hardware timer starts malfunctioning in the following manner. Instead of interrupting the CPU once every 10 time units (which is what the OS expects), it starts interrupting it every 5 time units (without the OS realizing this.) What problems is it likely to create for the CPU scheduler, if it is: (i) a proportionate-fair scheduler or (ii) a reservation-based scheduler? What would happen if the malfunction results in interrupts every 20 time units instead of every 10 time units?

(b) Consider the SFQ scheduler. How will the behavior of the scheduler change if we started scheduling processes in an increasing order of their finish tags (instead of their start tags)?

(a)
(i) Proportionate-fair scheduler → The proportionate fair scheduler allocates CPU to runnable processes in the ratio of their weights. Even though the CPU gets interrupted after 5 time units instead of 10, the proportionate-fair scheduler will be able to maintain fairness of CPU distribution amongst the processes. The only impact will be that a scheduling decision will be made every 5 time units instead of 10 (if the running process doesnt block) though this will remain hidden from the scheduler.

eg Thread A (W=1)

Actual Distribution Thred B (W=2)



The proportion is maintained though the absolute amount per scheduling decision is decreased i.e. instead of 10 units A gets scheduled out after 5 units

If interrupts occur after 20 time units, the proportion will still be maintained (fairness), the actual allocated time will increase.

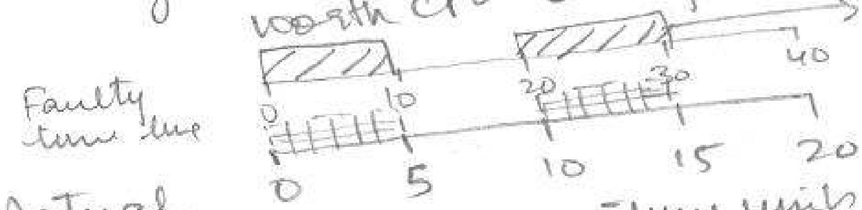
CPU scheduling ...

(ii) Reservation Based Scheduler → The RB scheduler gives absolute guarantees & is non work conserving

eg :- A(X,Y) → The process A is guaranteed X amount of CPU time every Y time units

In this case too the process will get X amount of CPU time every Y units though the scheduling will occur differently

eg :- A(10,20) → The RB scheduler guaranteed 10 time units worth CPU every 20 time units in CPU available to A

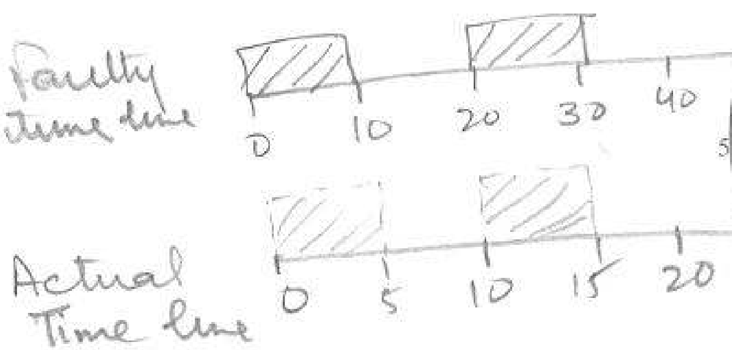


Thus in 20 time units A got 10 units of service though in a different manner.

Similarly even if the interrupt occurs after 20 time units, the guarantees would be met though in the above stated manner

Continuing the same example

Guarantees would still be met in the first case but may not be met in the second - depends on the (x,y) pair apps. ask for



8/10

A gets 10 units in 20 time units

æ<¨"Zá3Š!"qõ7o□Dæò^Žå+□Ã¤'5í□k□UW®Ãæ□z□Š)□ŠŸ^þ€íÇ*÷□î$.í□¬Øàù]ü÷Ÿóh´ùî¥-

(b) SFQ scheduler will degenerate to WFQ scheduler.

To compute finish tag the length of the quantum needs to be known apriori and scheduling according to finish tags will imply that this quantum be known beforehand. If for calculation max. quantum length is taken & the thread uses less than max. then it will not receive its fair share

Good, 5/5

;cŽžò°8□1/Ú±|\Î□T¥\R□Ë&i:·Ç□"8uh□□Z&ßO¶eq

3. **Memory management**

(5+10+15=30 pts)

(a) Discuss why Linux reserves a fixed portion of every address space for kernel addresses. Are there any downsides to this choice?

(b) Consider an operating system that implements paging-based virtual memory management. Suppose you wrote a program that malloc-ed an `int` and then looped infinitely. Suppose we started a new copy of this program every minute. Disregarding memory allocated for anything other than for these processes, how would the memory consumption of the computer evolve with time?

(c) Consider a typical multi-programming system with a buffer cache that the OS maintains for improving the efficiency of disk I/O. We want to implement a predictable virtual memory manager (with a well-defined objective such as: "maximize overall system throughput" or "share page fault rates proportionally among the processes"). One way of doing this is to maintain information that lets us estimate working sets of processes (such as the gLRU QPosition versus # accesses histogram) and use it to partition RAM among the processes. In this question, we would like to see your ideas on including the buffer cache into the predictable RAM partitioning problem.

Here are some high-level proposals. For each proposal, explain what you understand it to be and any thoughts you may have on how to implement it. (Extra credit: You are welcome to make your own proposals, if you have any.) Finally, compare these proposals with each other.

- *Proposal 0:* Keep a fixed number of frames reserved for the buffer cache. Use gLRU within the buffer cache.
- *Proposal 1:* Treat the buffer cache as RAM allocated to a separate (non-existent) process. Use gLRU within the buffer cache.
- *Proposal 2:* View blocks in the buffer cache as part of the address space of processes they belong to.

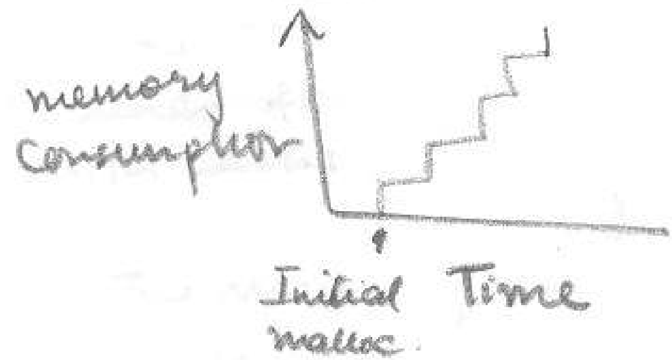
(a) Reserving a fixed portion of every address space for kernel addresses prevents TLB flush. Whenever the process changes from user mode to kernel mode & generates a particular virtual address of kernel no TLB flush occurs.

Good

5/5

The downside is that this kernel address space is fixed & if the kernel addresses have to increase then either the boundary has to be made dynamic i.e user space has to be eaten up or the kernel might crash.

(b) The memory consumption will increase with each malloc in a staircase manner



4/10

The firstmalloc() will get a chunk of memory & will allocate "int" as long as its available. When its finished it will get another chunk & so on, since there are no "free" calls and the programs keep looping thus the memory never gets released

CoW ⇒ no actual RAM allocated!

~~What will happen when the virtual address space gets exhausted?~~

Till when does this continue?

Proposal 0 →
(C) Similar to segmented queue

use GLRU here

Frames for
buffer cache

Comparison?

10/15

Proposal 1 → The memory manager will find it as another process & assign memory according to its objective. The process here can then implement its own memory manager, GLRU etc. (like Nemesis)

Proposal 2 → This amounts to extending the address space of a process. The portions of buffer cache become part of the process & then will be treated as other process pages.

how? What if buffer cache is assigned little mem?

Proposal 0 will improve disk I/O but may impact other processes as total RAM available to them will decrease. Proposal 1 may not improve I/O if all the pages for buffer cache get overwritten by other process pages. Proposal 2 gives processes the advantage of managing the buffer cache allocated to them by themselves

4. **I/O subsystem** (10+10+10=30 pts)

(a) Consider a computer running several backlogged TCP and UDP connections. Suppose we wanted to implement a proportionate-share (PS) scheduling algorithm to isolate these connections from each other. Where in the OS would you implement this? Draw parallels with a CPU scheduling system that is proportionally sharing cycles among several CPU-intensive and I/O-intensive processes. (Hint: Are certain connections like CPU-intensive processes while others are like I/O-intensive processes? In what ways?) Can you also think of any significant differences between PS scheduling for CPU and network bandwidth?

(b) Consider a computer with multiple Network Interface Cards (NICs.) Suppose we wanted to implement proportionate sharing (PS) of network bandwidth in such a system. Let us compare this with PS of CPU cycles in a multi-processor computer. What do you think are the counterparts of processes and processors in this setting? Do problems similar to "Weight Infeasibility" and "Short Jobs Arrival" that we studied for multi-processor CPU scheduling arise here? Explain your answer.

(c) Can you think of a resource for which being non-work-conserving can actually improve throughput? Make sure you clearly state what aspect of the operation of this resource you consider to be non-work-conserving.

(a) UDP connections can be compared to CPU intensive processes whereas TCP connections are like I/O intensive processes because there may be fragmentation in TCP & a packet may ~~have~~ to wait for all fragments before it can be further processed

CPU PS scheduler does not give extra advantage to I/O intensive processes. Whenever a blocked process is ready to run, the scheduler gives it the start tag same as virtual time (min of running threads start tags)

The CPU scheduler can account resources to the process/thread which consumed them i.e. process is the right abstraction for resource

7/10

principal whereas in n/w schedule the processing for a particular connection may involve multiple threads.

(b) NIC cards represent processors
Each socket (fd) is a process.

Yes the same problems are there in PS of n/w bandwidth.

If the weights assigned to each socket (application) are infeasible i.e. more bandwidth is assigned than NIC capacity then "weight infeasibility" will occur.

Nope

The "short jobs arrival" is also there. An application (socket) which receives packets intermittently may get the same bandwidth as a higher priority (more weight) process

ok

7/10

cŽžò‡8⌷1/Ú±¦\Î⌷T¥\R⌷Ë&i;·Ç⌷¨8uh⌷⌷Z&ßO¶eçt

(c) Disk is the resource for which being non-conserving can improve throughput. Data is spread on a disk & it may not exhibit spatial locality for a process. A proportionate scheduler may try to access data which is far away from the head. This will be an expensive operation whereas a non-work conserving scheduler instead of moving the head to a farther position will keep the head idle to improve throughput

10/10

If on the other hand, it is the latter, then that is a problem and you should tell me about when that happened.

5. **Feedback** (5+5=10 pts)

10/10

(a) Write two things you like about CSE 511.

(b) Write two things you dislike about CSE 511.

(a) Likes:

(1) The best thing I like is not accepting things written in papers as it is. We try to challenge the notions & try to bring out contradictions & flaws in the given arguments & accept things only when totally convinced

(2) The open discussion oriented atmosphere and not imposing a particular point of view

Thanks. I wish you had given a concrete example. Did you mean "jumping across topics" or "contradiction between things taught in one class and next"?

(b) Dislikes →

(1) Sometimes there is a mis-match in what was covered in previous class & what happens in the current class. Probably the flow of things can be improved

(2) Certain things have remained undiscussed like schedular activations

If it is the first, then it is because of my teaching 511 for the first time.