Notre Dame University

CSC 414 Applied Operating Systems
Final
Fall 2009

Duration: 2 Hours
Name:
ID:

1. (5pts) How can a single CPU system ”run” more than one ”program” at the same time?
by switching rapidly between programs

2. (5pts) Knowing that on a single CPU system only one instruction is executed at the same time why do we need to synchronize access to variables shared by more than one process?
because high level instructions are not atomic, they translate into a sequence of low level instructions which can be interrupted by context switch

3. (5pts) When setting up a paging system, explain why we need the page table size to be equal to the page size.
since pages are swapped as a whole we need a table to fill exactly one page

4. (5pts) What is the role of the Translation Lookaside Buffer (TLB) and how does it speedup memory references?
cache

5. (5pts) Compare interrupt driven I/O with polling. What is the advantage of using a Direct Memory Access (DMA) controller?. Do you think its efficiency is the same for data transfers of all sizes?
interrupt better than polling because the CPU can do some work while data is transferred between device and controller since device latency and transfer are much higher than CPU cycles. A DMA controller is efficient only for large transfers, i.e. bigger than it can fit in the device controller data buffer

6. In the readers-writers problem a shared buffer is accessed by multiple threads. When a writer thread is accessing the buffer no other writer or reader is allowed to access it. If a reader is accessing the buffer no writer is allowed to access it but other readers can.

   (a) (5pts) Use sempahores to solve the readers-writers problem.

```
reader                                   writer

wait(mutex);                             wait(wrt);
count++;
if(count==1)wait(wrt);                    access buffer
signal(mutex);
                                         signal(wrt);
access buffer

wait(mutex);
count--;
if(count==0)signal(wrt);
signal(mutex);
```

(b) (5pts) Is it possible for writers to starve in your solution presented in 6a?. Starvation occurs when the time a writer has to wait to enter its critical section is unbounded.
Yes because if readers get access to the buffer the writer has to wait for the last reader to finish which can be a long time if there is a steady stream of readers coming in

(c) (5pts) If you have answered yes in 6b then modify your solution such that when a writer is waiting, only the readers currently in their critical section are allowed to proceed. Any new reader is blocked until the waiting writer finishes accessing the buffer.

```
reader                                   writer

wait(noStarve);                         wait(noStarve);
signal(noStarve);
wait(mutex);                             wait(wrt);
count++;                                   signal(noStarve);
if(count==1)wait(wrt);                    access buffer
signal(mutex);
                                         signal(wrt);
access buffer

wait(mutex);
count--;
if(count==0)signal(wrt);
signal(mutex);
```

7. (5pts) A system has 5 frames. For each of the algorithms below give the number of pages faults for the reference string 1234573654632714. Show all your work.

(a) FIFO

(b) Optimal Algoritm

(c) LRU

1	2	3	4	5	7	3	6	5 4 6 3	2	7	1	4
1	1	1	1	1	7		7		7		7	7
	2	2	2	2	2		6		6		6	6
		3	3	3	3		3		2		2	2
			4	4	4		4		4		1	1
				5	5		5		5		5	4

1	2	3	4	5	7	3	6	5 4 6 3 2	7	1	4
1	1	1	1	1	7		6		7	7	
	2	2	2	2	2		2		2	1	
		3	3	3	3		3		3	3	
			4	4	4		4		4	4	
				5	5		5		5	5	

1	2	3	4	5	7	3	6	5 4 6 3	2	7	1	4
1	1	1	1	1	7		7		2	2	2	2
	2	2	2	2	2		6		6	6	6	4
		3	3	3	3		3		3	3	3	3
			4	4	4		4		4	4	1	1
				5	5		5		5	7	7	7

8. (10pts) A system has 5 processes and 4 resource types. Each resource has 10 instances. The initial state of the system is as shown in the table below.

	Alloc				Max			
	R_0	R_1	R_2	R_3	R_0	R_1	R_2	R_3
P_0	2	1	0	1	3	3	3	3
P_1	1	1	3	0	2	3	3	1
P_2	1	1	4	3	2	2	4	3
P_3	1	2	0	2	1	3	2	2
P_4	2	3	1	1	2	4	3	2

Use the Banker's algorithm to test whether the following requests are granted or denied, always starting from the initial state given above. Show all your work.

(a) Request (1,2,1,1) by P_0.
no safe sequence

(b) Request (1,1,2,1) by P_0.
2,0,1,3,4 is a safe sequence

(c) Request (1,1,1,1) by P_2.
request >need, request denied

9. (15pts)An inode in a 32-bit filesystem has 10 direct, 1 indirect, 1 double indirect, and 1 triple indirect pointers. The filesystem uses 1KB blocks on a disk that has 4000 tracks and 64 sectors per track. The size of the sector is 64 bytes and the first two tracks are used to store the inodes. Show all your work

(a) What is the maximum number of files that can be created.

(b) What is the maximum file size.

(c) If it takes 10 ms to read 1 block, how long does it take to read a file which has the maximum file size. Assume that the inode of the file is already cached in memory and that any block read from disk is cached in memory (i.e no block needs to be read twice from disk)

(d) How long does it take to read the same file if we use the linked list method.

Solution

(a) Total size of an inode=$4 \times (10+1+1+1) = 52$ bytes. Total size of the first two tracks=$2 \times 64 \times 64 = 8192$ bytes. Maximum number of inodes=$\frac{8192}{52} = 157$ =maximum number of files.

(b) A block stores 256 pointers therefore the maximum file size is: $10 + 256 + 256^2 + 256^3 = 16843018$ blocks$\approx 16GB$.

(c) In addition to the date blocks we need to read blocks containing pointers: 1 for indirect+(1+256) for double indirect+$(1 + 256 + 256^2)$ for triple indirect. Total blocks containing pointers=66051 blocks we add to that 16843018 data blocks to get a total of 16909069 blocks. Time to read the max file is 169090 seconds

(d) For the link list method each block contains 1024-4=1020 bytes of data therefore the total number of blocks to be read is $\frac{16843018 \times 1024}{1020}$

10. (15pts) Four processes q_1, q_2, q_3, q_4 are scheduled on a Linux system. q_1 starts at t=0 and later on creates (using fork()) q_2, q_2 creates q_3 and q_3 creates q_4, as shown below. q_1 writes to a buffer shared with q_2. q_2 issues a **non-blocking** read on the shared buffer. If no data is present in the buffer, read returns 0. Otherwise, it returns the number of bytes it read. Assume that for all processes the initial counter value is 6. The nice values for q_1, q_2, q_3 and q_4 are 0,0,0,-20 respectively. (Note that *fork()*, *write()* and *read()* are system calls).
Draw, with details, the Gantt chart for the system (include all necessary explanations).

q_1		q_2		$q3$		q_4	
inst	ticks	inst	ticks	inst	ticks	inst	ticks
fork()	1	fork()	1	fork()	1	do work	3
do work	5	a: x=read()	1	do work	7		
write()	1	if(x==0) goto a	1				
do work	1	do work	7				

11. (15pts) A 32-bit system has a 2-level page structure with 4kb page size. A process requests a 3MB data section and has been allocated two areas in memory, Area 1 is 1MB and Area 2 is 2MB. The virtual and physical memory layouts are shown in figure 1 below.

(a) Determine the number of valid entries in the directory table and their values.

(b) Set up the corresponding page tables. For each table specify the valid entries and their values.

(c) Suppose that Area 1 and Area 2 are swapped out to disk while the tables are still resident in memory. The process accesses , in sequence, all 1024 elements of an integer array. How many page faults does this operation cause? (Hint: the answer depends on whether the first element of the array starts at a page boundary or not)

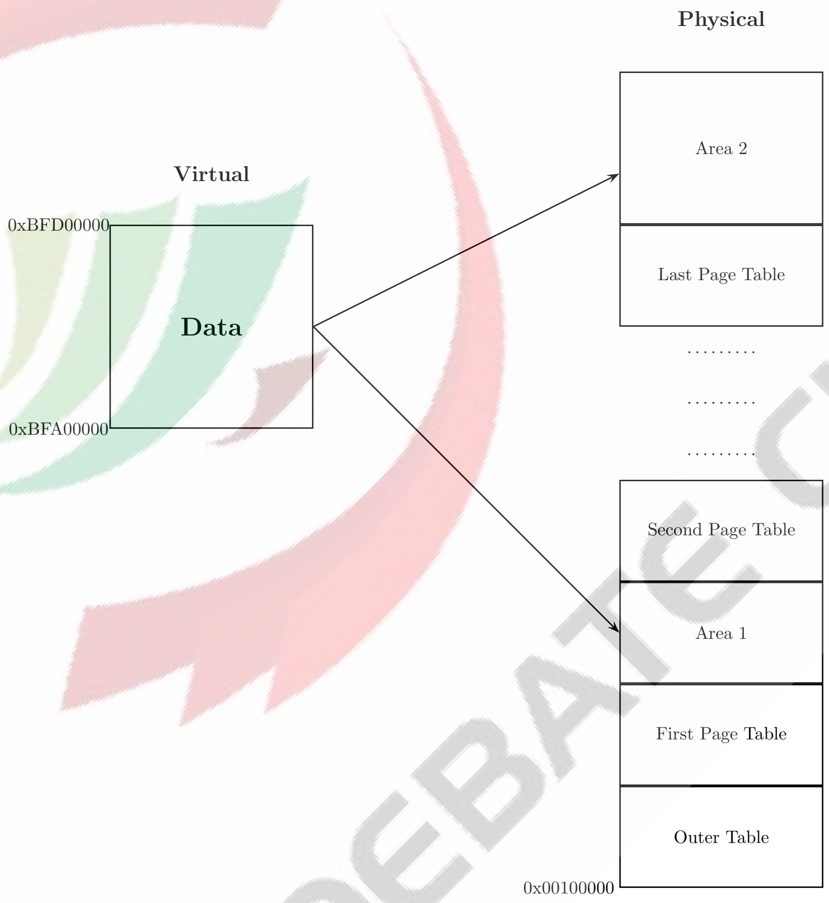
Virtual
Physical
0xBFD00000
Data
0xBFA00000
Area 2
Last Page Table
....
....
....
Second Page Table
Area 1
First Page Table
Outer Table
0x00100000

Figure 1: