操作系统习题（Part 2）

Basic Concept

- A user-level process cannot modify its own page table entries.
  - *If a user-level process was allowed to modify its own page table entries, then it could access physical memory being used by other processes or the OS kernel. Kernel mode is required to modify page table entries.*
- The scheduler is the part of an Operating System that determines the priority of each process.
  - ***FALSE**. The scheduler schedules processes based on user-specified priorities.*

Basic Concept

- Shortest Remaining Time First is the best preemptive scheduling algorithm that can be implemented in an Operating System.
  - ***FALSE****. SRTF cannot be implemented because it requires knowledge of the future.*
- The working set model is used to compute the *average* number of frames a job will need in order to run smoothly without causing thrashing.
  - ***FALSE****. The working set model is used to compute the minimum (total) number of frames a job will need in order to run smoothly without causing thrashing.*

Basic Concept

- Suppose we have a multithreaded process that accesses (reads and writes) to an external database, and it avoids race conditions by having each thread acquire a lock global variable before accessing the database. Using this approach, we can run two instances of this process accessing the same database at the same time.
  - ***FALSE****. A lock in one process instance will not affect the other process.*

Basic Concept

- If the Banker's algorithm will not approve a resource request, and the resource request is processed, then system necessarily will enter deadlock.
  - ***FALSE****. Banker's algorithm only guarantees that a system is in a safe state. If a system is in an unsafe state, deadlock may or may not occur.*
- In addition to greatly reducing the overall failure probability, moving from a single hard drive to using RAID 1 (mirroring) can also increase the throughput of small random reads.
  - ***TRUE****. The small reads need only be sent to a single disk, so in theory, a RAID-1 system can double small random read throughput.*

Basic Concept

- Remote Procedure Calls provide identical semantics to local calls.
  - ***FALSE****. RPCs can suffer from network/server failures, so the users need to deal with this difference.*
- mmap maps memory addresses to portions of files by changing page table entries to map virtual addresses to disk block addresses.
  - ***FALSE****. mmap() maps virtual memory addresses to pages in the buffer cache which holds the contents of a file.*

1. Effective Access Time

- Assume that the probability of page fault is 0.1%, memory access time is 100ns, and the average page fault service time is 25 ms, then the effective access time is:

100ns * 99.9% + (25*10^6ns + 100ns) * 0.1% = 25000ns = 25 us

1. Effective Access Time

Consider a demand-paging system with page tables stored in registers. When free frames are available, the time to process a page fault is 8 milliseconds. The cost of processing a page fault is still 8 milliseconds if the victim page chosen to be swapped out is never updated since it is loaded. However, if the victim page is dirty (meaning that it has been updated), it takes 20 milliseconds to process a page fault. Suppose the cost of each memory access is 100 nanoseconds, and that 70% of the victims are dirty. What is the maximum allowed page fault ratio to ensure an effective access time of 200 nanoseconds? (1 millisecond = 1 million nanoseconds)

$$(1-P)*100ns + P*(1-70\%)*(8ms+ 100ns) + P* 70\%*(20ms + 100ns) = 200ns$$

Effective Access Time

- Consider a memory system with a cache access time of 10ns and a memory access time of 110ns – assume the memory access time includes the time to check the cache. If the effective access time is 10% greater than the cache access time, what is the hit ratio *H*?

$$\textit{Effective Access Time} = H*T_{cache} + (1-H) * T_{memory}$$
$$1.1 * T_{cache} = H*T_{cache} + (1-H) * T_{memory}$$
$$1.1 \times 10 = H*10 + (1-H)110$$
$$11 = H*10 + 110 - 110*H$$
$$-99 = -100*H$$
$$H = 99/100$$

Page Size

- Consider a paging system that maps logical address space of 8 pages with 1024 bytes each page to a physical memory of 32 frames, the logical address is of ___ and the physical address is of ___.

A. 10 bits, 5 bits
B. 3 bits, 15 bits
C. 13 bits ,5 bits
D. 13 bits, 15 bits

Page Table

- Consider a machine with a physical memory of 8 GB, a page size of 8 KB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page? Be explicit in your explanation.

Since each PTE is 4 bytes and each page contains 8KB, then a one-page page table would point to 2048 or 2_{11} pages, addressing a total of $2_{11} * 2_{13} = 2_{24}$ bytes. Continuing this process:

Depth	*Address Space*
1	2_{24} *bytes*
2	2_{35}*bytes*
3	2_{46}*bytes*

Page Table

- In a 32-bit Linux, how much physical memory is needed for a process with three pages of virtual memory (for example, one code, one data, and one stack page)?
  - *Six physical pages totaling 48KB are needed: one for the first-level page table, one for one page of the second-level page table, one for one page of the third- level page table, and three for the process' three pages.*

Synchronization

- A common parallel programming pattern is to perform processing in a sequence of parallel stages: all threads work independently during each stage, but they must synchronize at the end of each stage at a synchronization point called a ***barrier***. If a thread reaches the barrier before all other threads have arrived, it waits. When all threads reach the barrier, they are notified and can begin the execution on the next phase of the computation.

Example:

```
while (true) {
Compute stuff;
BARRIER();
Read other threads results;
}
```

Synchronization

```
class Barrier() {
        int numWaiting = 0; // Initially, no one at barrier
        int numExpected = 0; // Initially, no one expected
        Lock L = new Lock();
        ConditionVar CV = new ConditionVar();
        void threadCreated() { L.acquire();
                numExpected++;
                L.release(); }
        void enterBarrier() { L.acquire();
                numWaiting++;
                if (numExpected == numWaiting) { // If we are the last
                numWaiting = 0; // Reset barrier and wake threads
                ? ? ? ?; // Fill me in
                } else { // Else, put me to sleep
                ? ? ? ?; ____ //Fill me in
                }
        L.release() ; }
}
```

Synchronization

- Three processes are synchronizing on a shared code segment which is protected by a semaphore. If at most two processes are allowed to enter the code segment simultaneously, which of the following results shows the possible values that the semaphore may have?

A. 2, 1, 0, -1 B. 3, 2, 1, 0 C. 2, 1, 0, -1, -2 D. 1, 0, -1, -2

Thread

- In Many-to-One thread model, What would happen if a thread in a multithreaded process makes a blocking system call ?
  - A. the other threads of the process would continue to run
  - B. the entire process would get blocked
  - C. the blocking thread will get killed
  - D. the blocking thread will never get to run again

Synchronization

- Consider the following three threads in a concurrent program that uses semaphores Sem1, Sem2, and Sem3.

Thread 1 L1: **sema_down(Sem3);** **print("2");** **sema_up(Sem2);** **goto L1;**	**Thread 2 L2:** **sema_down(Sem1);** **print("6");** **sema_up(Sem3);** **goto L2;**	**Thread 3 L3:** **sema_down(Sem2);** **print("1");** **sema_up(Sem1);** **goto L3;**

Synchronization

- Are there initial values that can be given to the semaphores so that the threads cooperate to print a string that begins with 1621621621621 6? If so, give the initial values

Yes. Initialize the semaphores as follows:
Sem1=0, Sem2=1, Sem3=0

Thread

```
void *thread_start(void *arg)
{
        thread_count++;
        if (thread_count == 3) {
                char *argv[] = {"/bin/ls", NULL};
        execv(*argv, argv);
        }
        printf("Thread: %d\n", thread_count);
        return NULL;
}

int main(int argc, char *argv[])
{
        int i;
        for (i = 0; i < 10; i++) {
                pthread_t *thread = malloc(sizeof(pthread_t));
                pthread_create(thread, NULL, &thread_start, NULL);
                pthread_join(*thread, NULL);
        }
        return 0;
}
```

Thread

- When you run the program, what will be the output (assume that all system calls succeed)?
  - *Thread: 1 Thread: 2 (contents of current directory)*

- If we removed the line pthread_join(*thread, NULL), could the output change?
  - *Thread: {1...10} (contents of current directory zero or one times)*

CPU Scheduling

- Consider the following **single-threaded** processes, arrival times, and CPU processing requirements:

Process ID (PID)	Arrival Time	Processing Time
1	0	6
2	2	4
3	3	5
4	6	2

CPU Scheduling

- For each scheduling algorithm, fill in the table with the ID of the process that is running on the CPU. Each row corresponds to a time unit.

•Fo

•W 2 that arri

• If

Time	FIFO	RR	SJF
0	1	1	1
1	1	1	1
2	1	1	1
3	1	2	1
4	1	1	1
5	1	3	1
6	2	2	4
7	2	1	4
8	2	3	2
9	2	4	2
10	3	2	2
11	3	1	2
12	3	3	3
13	3	4	3
14	3	2	3
15	4	3	3
16	4	3	3

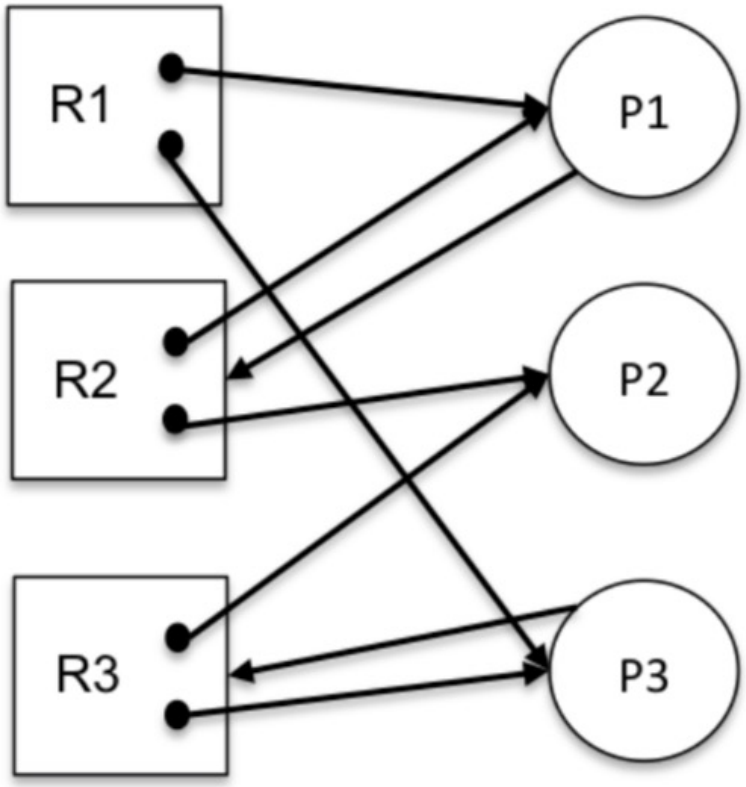
CPU Scheduling

- Calculate the response times of individual processes for each of the scheduling algorithms. The response time is defined as the time a process takes to complete after it arrives.

	PID 1	PID 2	PID 3	PID 4
FIFO	6	8	12	11
RR	12	13	14	8
SJF	6	10	14	2

Deadlocks

- Consider the following resource allocation graph:



1.Does the above allocation graph contain a deadlock?

1.No. A possible execution sequence is P2, P1, P3.

2 Assume now that P2 also demands resource R1. Does this allocation graph contain a deadlock?

2Yes. If P2 demands R1, none of P1, P2, or P3 will get all resource they need to be processed

Inverted Page Tables

- Assume we have a simple, demand paging environment, with no segmentation. Processes P1 and P2 both have logical memory addresses in the range 0 . . . 99, inclusive. The page size is 5.
- The hash table portion of the structure uses a hash function which simply calculates the index by adding the numerical portion of the process identifier and the logical page number. Note that this is a really bad hash function because it requires a large hash table, but for our exercise, it is sufficient. The hash table and the inverted page table shown are below.

Inverted Page Tables

Hashtable

Index	Process ID / logical page	Frame ID
…	…	…
3		
4	P2 2	1
5	P1 3	4
…	…	…
18	P1 17	2
19	P1 18	3
…	…	…
30		

Inverted Page Table

Index	Logical address info (minimized for simplicity)
0	
1	P2:RW
2	P1:RW
3	P1:R
4	P1:RW

Inverted Page Tables

- Calculate the physical addresses for the following logical addresses where the number after the colon is the logical address (**not the logical page**). Assume the first page of a process is page number 0, and that a mechanism exists to find a free frame in memory

P1:17

Logical page 17/5 = 3 offset = 2. hash table index = 3 + 1, follow chain to entry 5. From lookup, physic frame = 4. Address = (frame)∗(frame size) + offset = 4 ∗ 5 + 2 = 22

P1:92

Logical page = 18. offset = 2. Hash table index = 19. From lookup logical frame = 3. Address = 17

Inverted Page Tables

- When process P2 attempts to read logical address 97, what happens? Specifically, describe the changes in the data structures, and what the process perceives of these changes.

This causes a page fault. Frame 0 is allocated, and the entry in the IPT is updated with the logical address information for P2 frame 19. Next the hashtable entry at index 21 (19 + 2) is mapped from P2:19 to frame 0. After this processing is complete, the P2 resumes execution as before.

TLB Optimization

- Since high-performance scientific computing applications run on very expensive supercomputers, it is very important to optimize their performance. Consider the following scientific computing function:

```
MatrixAdd(int a[][], int b[][], int n)
{
        for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
        a[j][i] += b[j][i];
}
MatrixAllocate(int n)
{
        a = malloc (n * size of *a);
        for (i = 0; i < n; i++)
                a[i] = malloc(n * sizeof a[i]);
        return a;
}
```

TLB Optimization

- You observe that the program runs more slowly than expected when calling MatrixAdd, so you turn on profiling and notice that the function is causing large numbers of TLB flush operations – much more than expected. Explain the likely cause and implement a simple fix. Also, explain what you generally expect to happen to paging performance when your fix is implemented?

File System

- What is the maximum file size supported by a file system with 16 direct blocks, single, double, and triple indirection? The block size is 512 bytes. Disk block numbers can be stored in 4 bytes.

Number of blocks:
- Direct Blocks = 16 blocks
- Single Indirect Blocks = 512 / 4 = 128 blocks
- Double Indirect Blocks = 128 * 128 = 16384 blocks
- Triple Indirect Blocks = 128 * 128 * 128 = 2097152 blocks

Total number of blocks = direct + single + double + triple = 16 + 128 + 16384 + 2097152 = 2113680 Total number of bytes = 2113680 * 512 = 1.08220416 E 9 = 1.08 GB

File System

- Consider a UNIX filesystem with the following components:
  - Disk blocks are 4096 bytes. Sectors are 512 bytes long.
  - All metadata pointers are 32-bits long.
  - An inode has 12 direct block pointers, one indirect block pointer and one double-indirect block pointer.
  - The total inode size is 256 bytes.
  - Both indirect and double indirect blocks take up an entire disk block.

File System

- How much disk space, including metadata and data blocks, is needed to store a 4 GB DVD image file? You can leave the answer in symbolic (e.g., 6MB + 3KB) form

Requires 2^{32} bytes of actual data: 1048576 data blocks.
1024 block ptrs per indirect block.
1,048,564 blocks/1024 = 1024 indirect blocks needed.
1 double-indirect block needed.
256 bytes for inode.
Total: 4,299,165,952 bytes =
*4Gb+256b+1*4096b+1024*4096b+1024*1024*4096b*

Disk

- Suppose that we have a disk with the following parameters:
  - 1TB in size
  - 7200 RPM, Data transfer rate of 40 Mbytes/s (40×10^6 bytes/sec)
  - Average seek time of 6ms
  - ATA Controller with 2ms controller initiation time
  - A block size of 4Kbytes (4096 bytes)

Disk

- What is the average time to read a random block from the disk (assuming no queueing at the controller).

$$T_{read} = Controller + Seek + Rotational + Xfer\ Time$$

$$= 2ms + 6ms + \frac{1}{2}\left(\frac{60000\frac{ms}{min}}{7200\frac{revolutions}{min}}\right) + \frac{4096bytes}{\left(40\times10^{6}\frac{bytes}{s}\right)\times\left(10^{-3}\frac{s}{ms}\right)} = 12.269\ ms$$

$$= 12.269ms$$

$$T_{read} = 12.269ms$$

RAID

- Consider that many RAID devices now ship with the following options:
  - RAID 0 - data striped across all disks
    RAID 1 - each disk mirrored
    RAID 5 - striped parity
- Assume a system with 8 disks
- For each level, how much usable storage does the system receive?

RAID 0 – 8 disks
RAID 1 – 4 disks
RAID 5 – 7 disks

RAID

- Assume a workload consisting only of small reads, evenly distributed. Assuming that no verification is performed on reads, what is the throughput of each level assuming one disk does 100 reads/sec?

RAID 0 – 800 reqs/sec
RAID 1 – 800 reqs/sec – reads can be satisfied from both disks in a pair
RAID 5 – 800 reqs/sec – no need to read the parity, so no loss of read performance, only space

RAID

- Assume a workload consisting only of small writes, evenly distributed. Again, calculate the throughput assuming one disk does 100 writes/sec

RAID 0 – 800 reqs/sec
RAID 1 – 400 reqs/sec – need to write to both disks in a pair
RAID 5 – 100 reqs/sec if you read all of the disks to recalculate the parity

RAID

- For each level, what is the minimum number of disks that may fail before data **may** be lost?

RAID 0 – 1, but data loss is guaranteed at the first lost disk
RAID 1 – 2, if you happen to lose both disks in a pair
RAID 5 – 2, but data loss is guaranteed on the second disk

RAID

- For each level, what is the minimum number of disks that must fail to **guarantee** data loss?

RAID 0 - 1
RAID 1 – 5, if you happen to get really lucky and lose one from each pair before losing the 5th
RAID 5 - 2