SJF = Shortest Job First

NUMA = Non-Uniform-Memory-Access

FIFO = First In First Out

FCFS = First Come First Serve

PTBR = Page table base register

FCB = File control block

ACL = access control list

PTLR = page table length/limit register

TLB = translation lookaside buffer

PCB = process control bock

LFU = least frequently used

COW = copy on write

VFS = virtual file system

MMU = memory management unit

FAT = file allocation table

STBR = segment table base register

MFU = most frequently used

STLR = segment table limit register

LRU = least recently used

EAT = effective access time

Super block is place where metadata for file system is stored

Page fault is an interrupt caused by a referred page not being loaded in the memory

Thrashing is when all processes that are working in memory are larger than the physical memory available

Internal fragmentation is the allocated memory that is unused

Range-lock is a file lock that covers a range of bytes in a file

Advisory lock – lock that is not tightly enforced by the OS

Frame is a used in paging and is a unit of allocation for physical memory

External fragmentation is memory holes left in the free list after allocation

Monitor is a construct used to protect critical sections.

Starvation is when a process doesn’t get the CPU for long periods of time

File descriptor is the curses that indicates the position of a file being read/written to by a process

Windows-native file systems employ ACLs for protection/sharing

The data block free list for a file system can usually be found in the superblock

Peterson’s solution is a process synchronization that doesn’t require hardware support

Swap instruction, peterson’s solution, and test-and-set instruction are process synchronizations that involve the busy wait

Local page replacement schemes: when a process needs a frame It can send out, any other page that process may have in memory

File operations: Create, Write, Delete, Open, Close

Conditions for deadlock: mutual exclusion, circular wait, hold and wait

Dirty bit in page tables indicates when a file has been modified

TLB miss happens when required page to frame isn’t present in TLB

Bealdy’s anomaly occurs when an increased number of frames leads to a higher page-fault rate

File-system metadata includes inode-free-list and data block free list

Main disadvantage of multi-level page tables is higher page access time during TLB misses

Typical operations on directories: create file, delete file, rename file

A successful fork return value for a child is 0

Trap switches from user mode to kernel mode

POSIX calls include fork, kill, wait, getpid, getppid

SJF Premptive can pause a process currently being worked on to work on a shorter one. Non-preemptive will work on the process until it’s finished

32 bit memory space with 8kB pages per frame and a two page level table (13 bits offset). Max size of page table in bytes needed for a process with at least 4MB of data at the start of its virtual address range. Assume each page table entry has 32 bits.

Mex memory with 32 bits is 2^32 bytes

Index page of 4 kilobytes needs n bits. Break it down into 4\*1000 or 2+10. 12 bits

Then you subtract the two numbers (32-12) to get 20 bits.

Since there are two levels, half of the page table index bits are used to index each level of the table. First level always allocated for 4\*20^10 bytes.

Second level needs enough tables to span 4MB. Since each page is 4kB, you need (4MB/4kB) 1K entries with each entry being 4 bytes. We need 4KB for second level.

4kB + 4kB = 8 kB total

Best fit memory allocation approach and principle:

Scan the free list for the block that is the smallest available and fits the request. It’s to leave the smallest fragment behind so we have the least waste in external fragmentation.

**SHORT CONCEPTS- S13**

Starvation is: \_when a process doesn’t receive CPU in extended periods of time\_

A Monitor is: \_\_a language construct to protect critical sections, includes a set of functions that must be executed mutually exclusively \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

External fragmentation is: \_memory holes left in the free list after allocation \_\_\_\_\_

A frame is: \_\_\_a unit of allocation for physical memory when using paging \_\_\_\_

An advisory lock is: \_\_\_a lock that is not strictly enforced by the OS\_\_\_\_\_\_\_\_\_\_\_

A range-lock is: \_\_\_a type of file lock covering a a range of bytes in a file\_\_\_\_\_\_\_

Internal fragmentation is: \_\_\_unused memory in allocated memory\_\_\_\_\_\_\_\_\_\_\_\_

Thrashing is: \_\_ when the working set of all processes in memory is larger than the physical memory available \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A page fault is: \_\_ when a referred page is not loaded into memory \_\_\_\_\_\_\_\_\_\_\_

The super-block is: \_\_the part of a file system where we store metadata for the file system itself\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Multiple Choice**

The cursor that indicates the position in a file being read/written to by a process is stored in: the file descriptor

Windows-native file systems employ the following for file protection/sharing: ACL’s

The data block free list for a file system can usually be found in: the superblock

The following is a process synchronization solution that does not require hardware support:Peterson’s Solution

The following process synchronization solutions involve busy wait: All of the Above

In local page replacement schemes, when a process needs a frame it can page out: Any other page that process may have in memory

If you are using a single-processor CPU system and want to protect a short critical section, the best and most efficient choice will be: disabling interrupts

The optimal page replacement algorithm chooses the page to be replaced as follows:

The page that will be referenced the farthest in the future (or not referred at all);

The best technique for file block allocation when we intend to use random access is: Index Allocation

What page would be the worst candidate to replace out of the following ones during the execution of page replacement algorithm: A page that has the referenced bit set to 1 and the dirty bit set to 1;

**FILL IN THE BLANKS**

Mention three file operations: Create, Read, Write, Reposition, Delete,

Truncate, Open, Close

Mention two required conditions for deadlocks: Mutual Exclusion, Hold and Wait, Non-preemption, Circular Wait

The dirty/modified bit in page tables is used to indicate: page has been

modified/written to

A TLB miss occurs when: The required page to frame is not present in the TLB

The Belady’s anomaly occurs when: An increased number of frames leads to a

higher page-fault rate

File-system metadata (not file metadata) includes the following two items: inode-free-list , and data block free list

The main disadvantage of multi-level page tables when compared with flat page

tables is : higher page access time during TLB misses

Mention two typical operations on directories: Search for file, list directory, Create file, Rename File, Delete file, traverse file system

When successful, the return value of fork() in a child process is: zero

A trap switches the system from user mode to kernel mode.

**Short Answer Section**

51- Assume a 32 bit memory space with 8Kbytes pages/frames and a two level page table (13 bits of offset). Answer the following: what is the maximum size of the page table in bytes needed for a process that has 8[MB] of code at the start of its virtual address range? Assume each page table entry has 32 bits.

Max Memory addressable with 32 bits = 232 [bytes]

Size of page table = Max Memory addressable with 32 bits / Page Size

Size of page table = 232 [bytes] / 8[Kbytes] = 232 [bytes] / 213 [bytes] = 219[entries] Since each entry has 4 bytes we have􏰀Size of page table = 4\*219[entries] = 2 [MB]

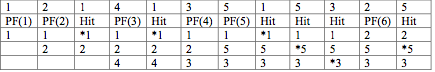
To index pages of 4[Kbytes] you need n bits such that:

Offset 2n= 8[Kbytes]=8\*1024[bytes]=8192[bytes]􏰀n = log2(8192) = 13 [bits] The rest of the bits in the address are used to index the page table 32-12=19[bits]

Explain the best fit memory allocation approach and the principle behind it.

In this algorithm you scan the free list for the block that is the smallest available and that fits your request. The rationale is to leave the smallest fragment behind so we have the least waste in external fragmentation.

53- Determine how many page faults will occur with the following reference string and assuming the use of the *Optimal* page replacement algorithm with 3 frames of memory? {1, 2, 1, 4, 1, 3, 5, 1, 5, 3, 2, 5} Display the final set of pages in memory at the end. (Hint: Assume physical memory is originally empty.)



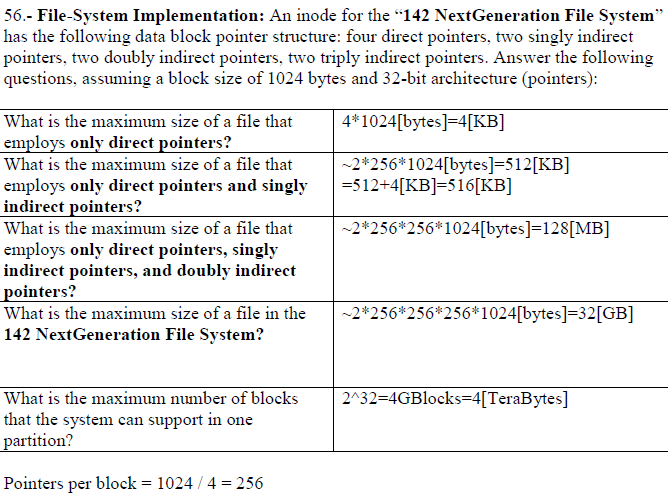
54.-What is the working set of a process that has the following reference string? {1, 2, 6, 4, 1, 2, 5, 6, 2, 3, 4, 5}

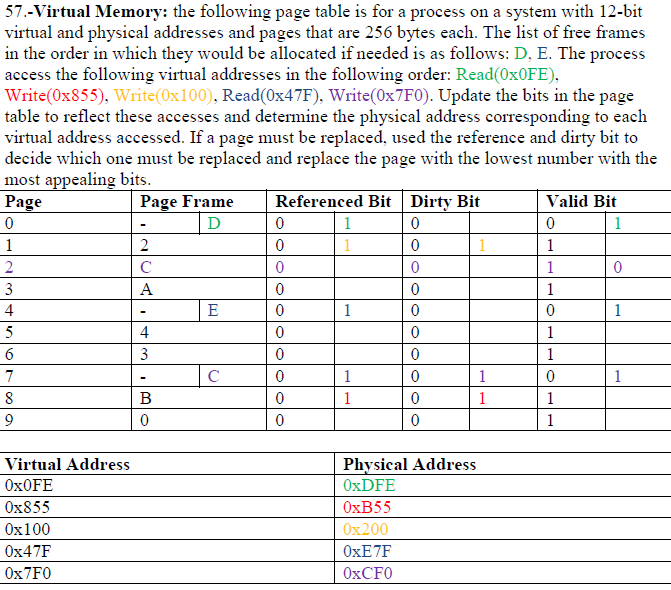
Working Set = {all pages that are used} = {1, 2, 3, 4, 5, 6}

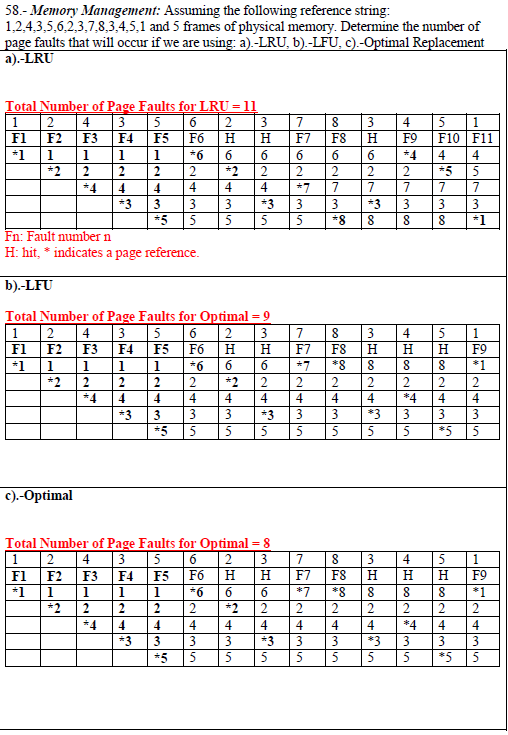
55.File “temp” is composed of seven disk physical blocks namely 5, 8, 4, 13, 11, 9, 17 and 10 (in that order) in a FAT-16 file system. Fill the FAT-16 table below with the representation for such file (Hint: End of file is represented as -1):



**LONG ANSWER - ESSAY QUESTIONS**







59.- Process Synchronization: Assume a very large array of size n containing words in which we want to search for a given word W and figure how many occurrences of the word are in the array. Given you have a multiprocessor with 4 processors how would you implement a multi-process/multithreaded program to find the number of matches for W in the array in the minimum time? Use only shared variables and semaphores as needed. Give all data structures used, its initial values and the pseudo code for all the processes or threads you need.

Solution draft: a main thread creates 4 children threads that can search the array in parallel and determine the number of matches in ¼ of the array each. The children then pass their number of matches found to the main thread that can then aggregate them and present the user with a final match count of the word W in the array in ¼ of the time that it would take to do all in a single threaded program. The children notify the main thread when they are done by using semaphores.

Assumptions: array[] is filled and not being modified, and n % 4 = 0.

int matches[4]={0};

Semaphore child2parent[4]={0};

Main\_thread(){

int totalMatches=0;

for(int i=0; i < 4; i++) {

thread\_create(childThread, i);

}

for(int k=0; k < 4; k++){

wait(child2parent[k]);

totalMatches += matches[k];

}

printf(“Total Matches=%d\n”, totalMatches);

}

childThtread(int i){

for(int j=i\*(n/4); j < (i+1)\*(n/4); j++) {

if(array[j] == W)

matches[i]++

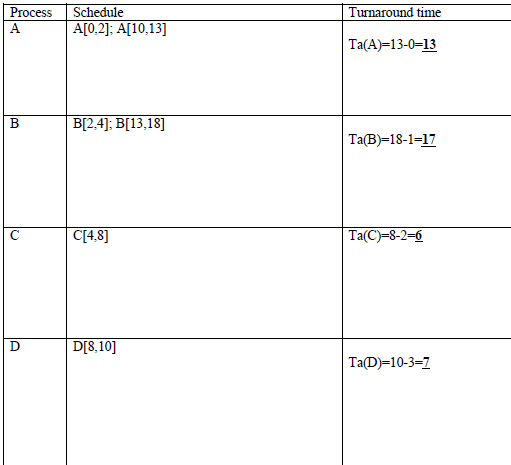
}

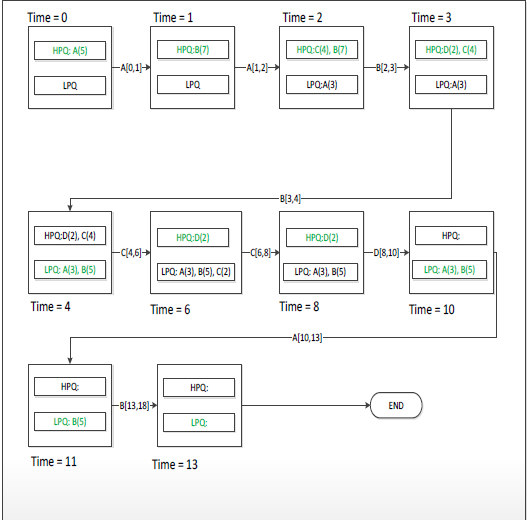
// This thread work is done, tell parent it is OK to use matches[i]

signal(child2parent[

60.- Process Scheduling:The notation (P, y, z) means that process P arrives at time y and needs z time units of processing time. Assume a scheduler observes the following arrivals: (A, 0, 5); (B, 1, 7), (C, 2, 4), (D, 3, 2)

Assume the context switch to a different process is 0 time unit. Provide the resulting schedule chart, turnaround time for each process. Assume the scheduler implements a multilevel queue system with the following restrictions: there are two queues: processes enter the system in the high-priority queue and they are served in round robin fashion (time slice = 2) while there for only one time slice and then moved to the next queue which is served using SJF (non-preemptive). The scheduler also distributes the time as follows between the two queues: 6 time units for the highest priority queue and 2 time units for the lowest priority queue starting with the highest priority queue at time 0, if there are no jobs in one queue the scheduler switches to the next queue right away.

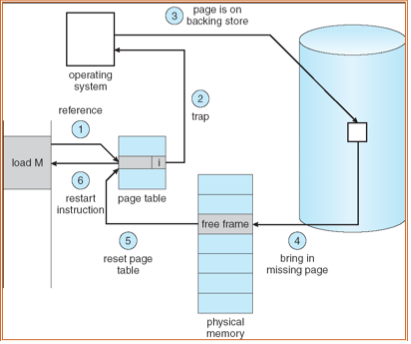




Memory Management

What is a page fault and what are the events and actions associated with it in modern operating systems like Linux. Be specific as to describe what happens with the process that incurs the page fault.

A page fault is the event that occurs when a process reference a page that does not exists in physical memory. If the virtual memory referenced does not exist, the process is generally killed; if the page is hosted on disk then it gets read into a free frame.



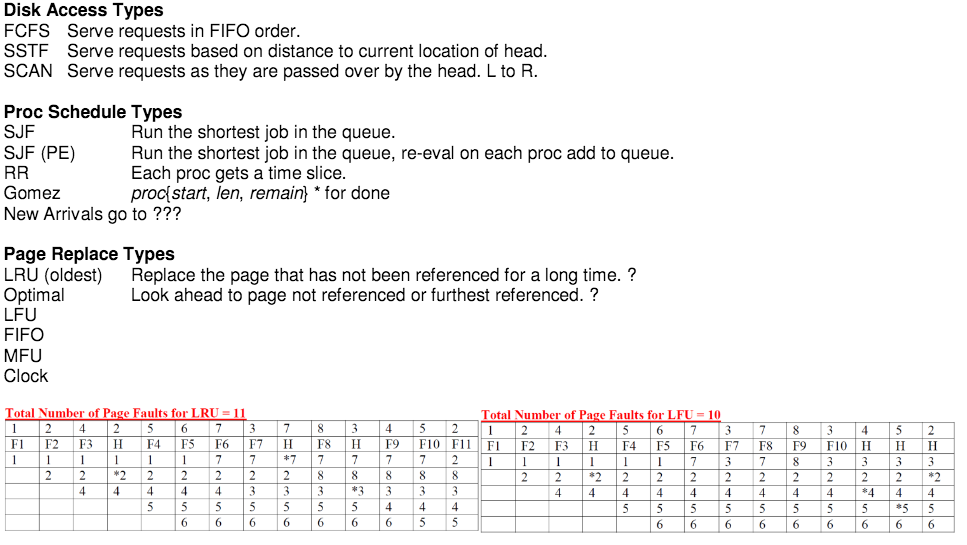
Not shown in the figure

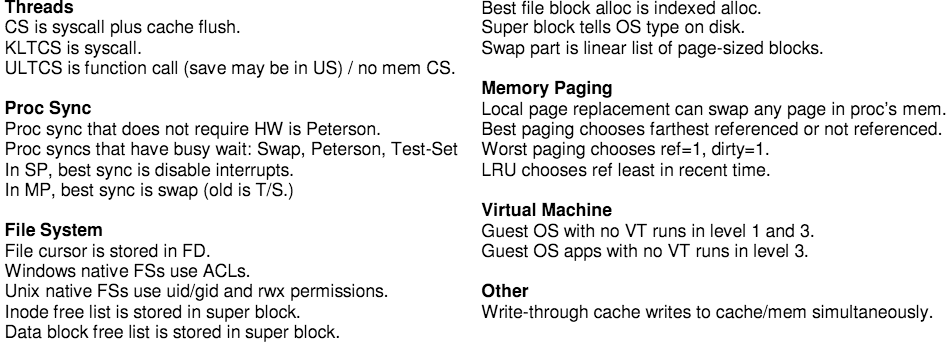
If page does not exist, signal the process to die perhaps creating a core file.

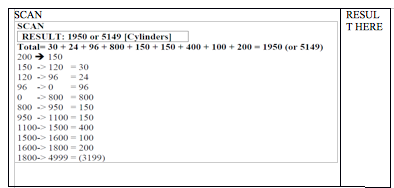
If page in backing store, place process in device queue for device hosting the page.

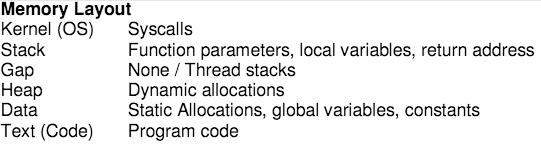
Place process in ready queue again

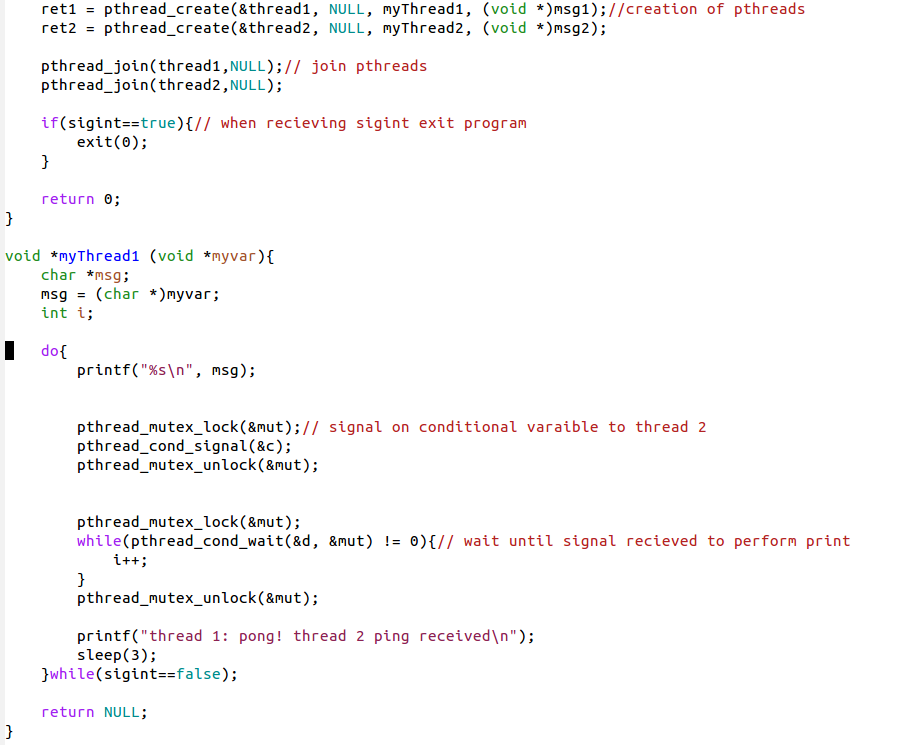
PICS



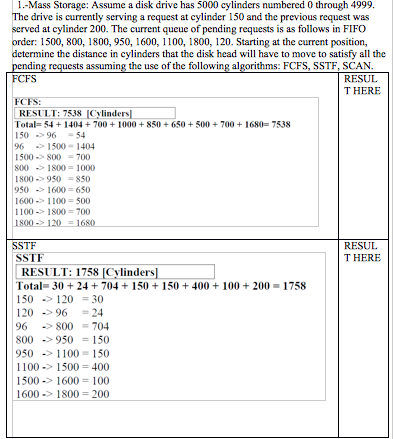


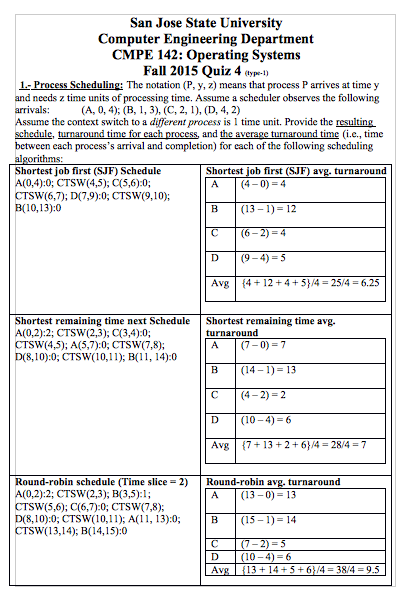


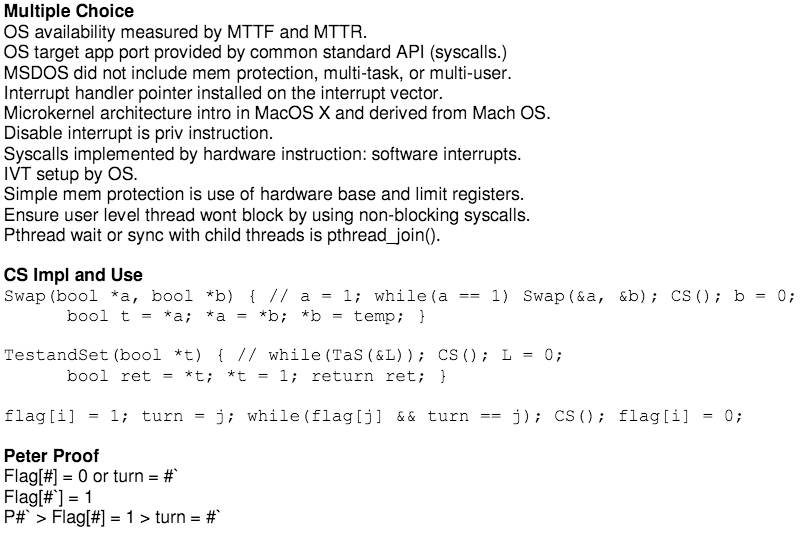


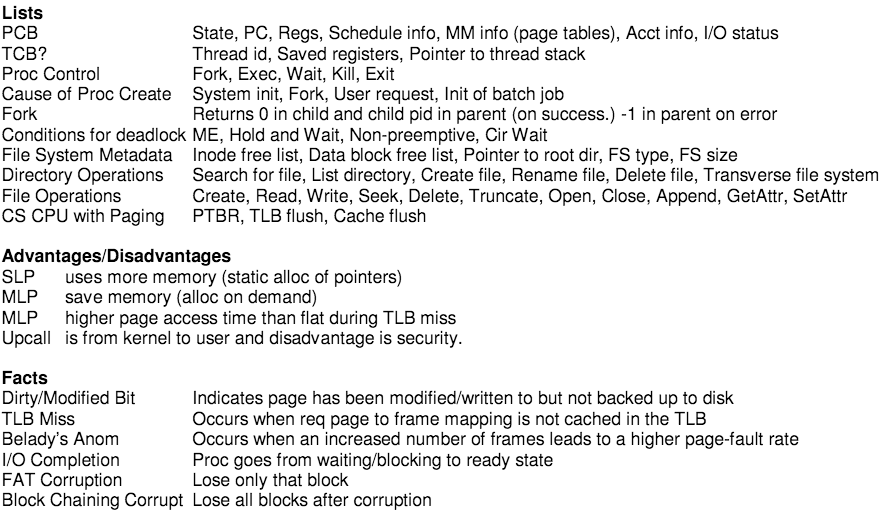


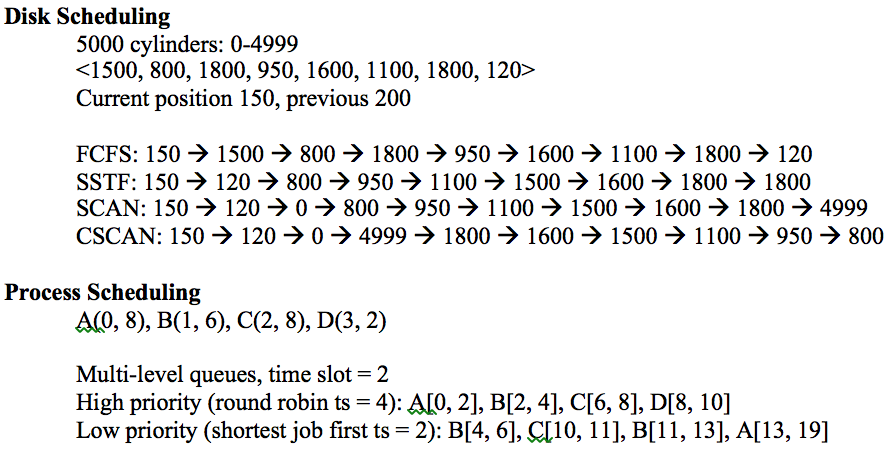
DEADLOCK CODE

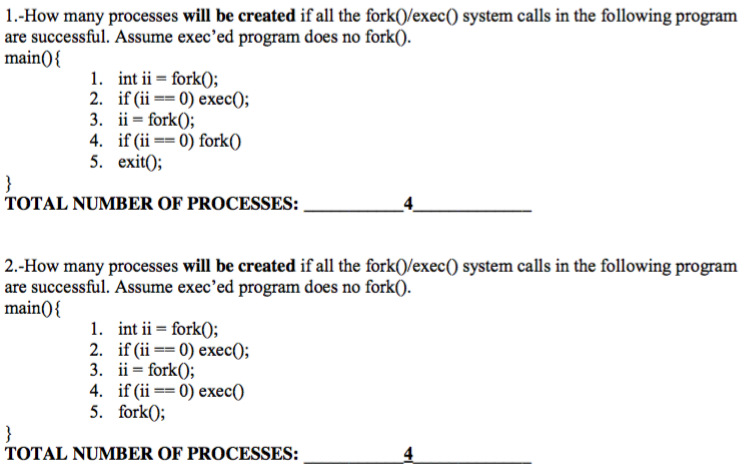


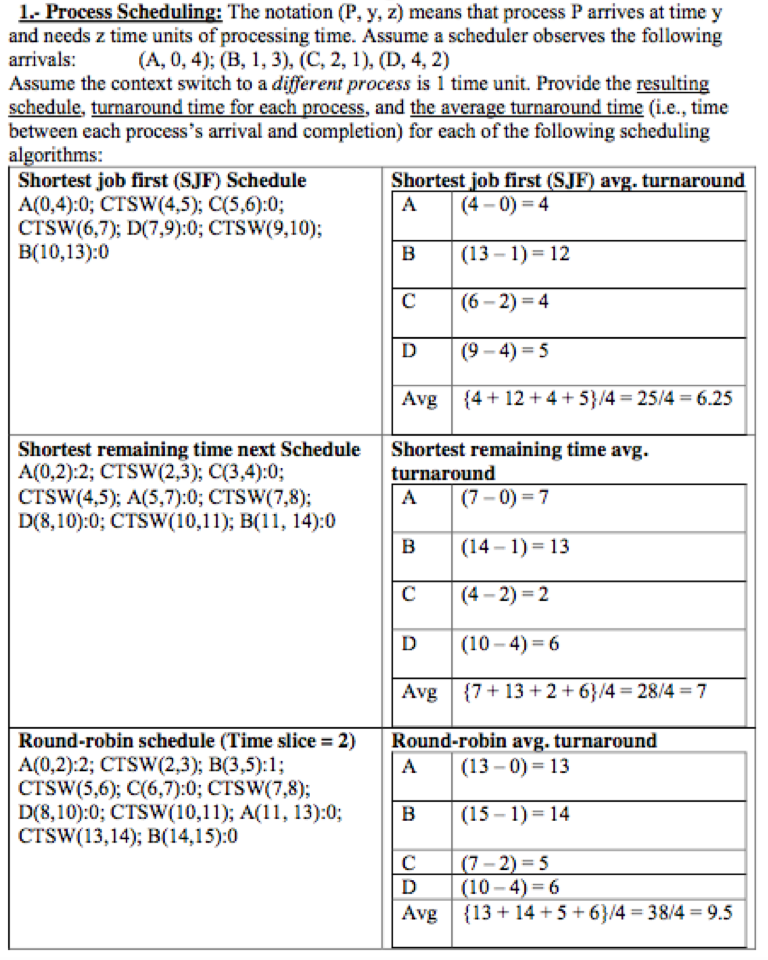


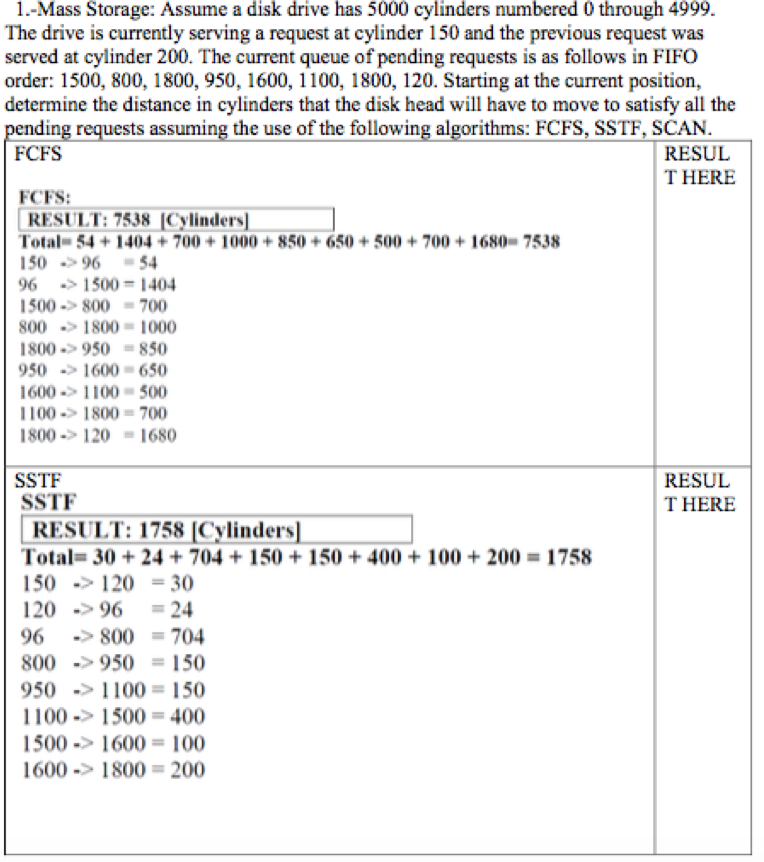


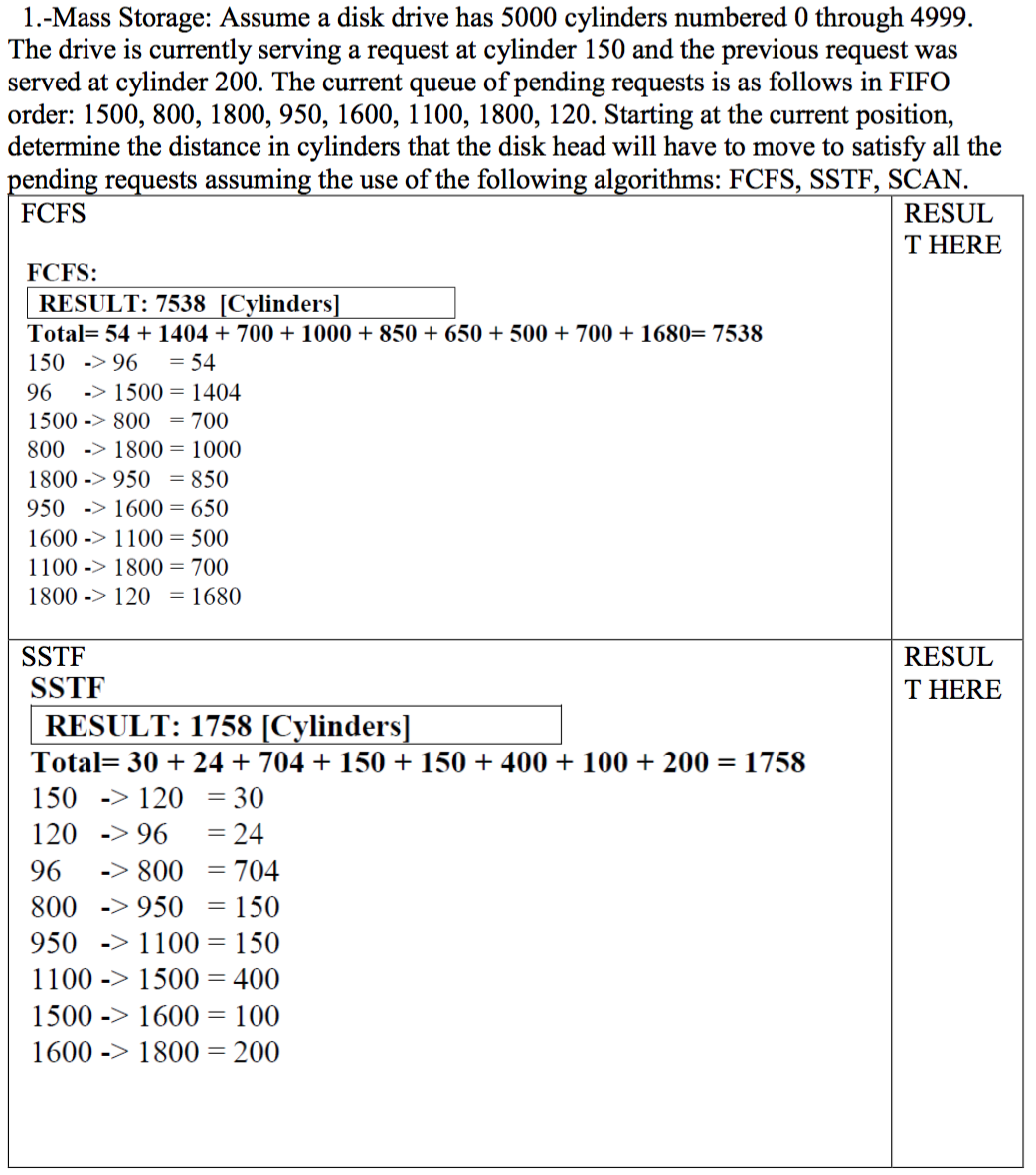


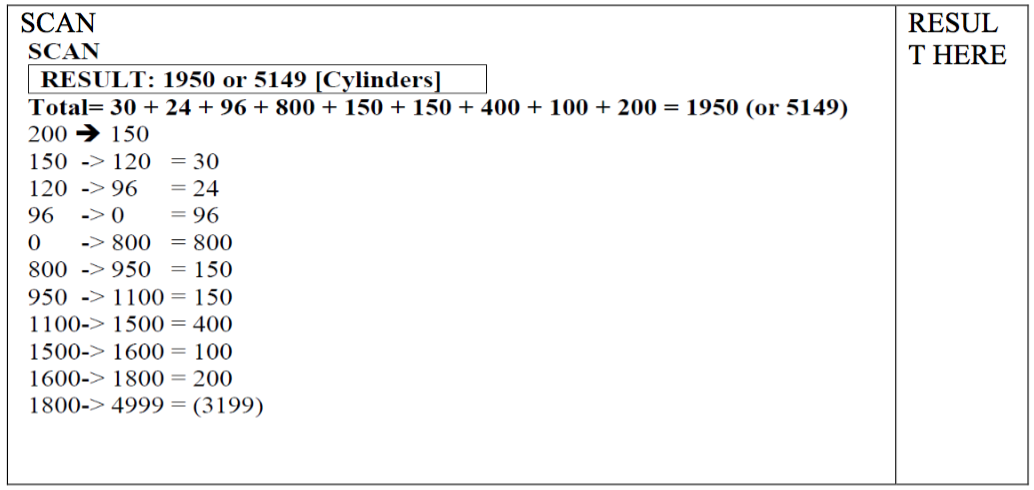


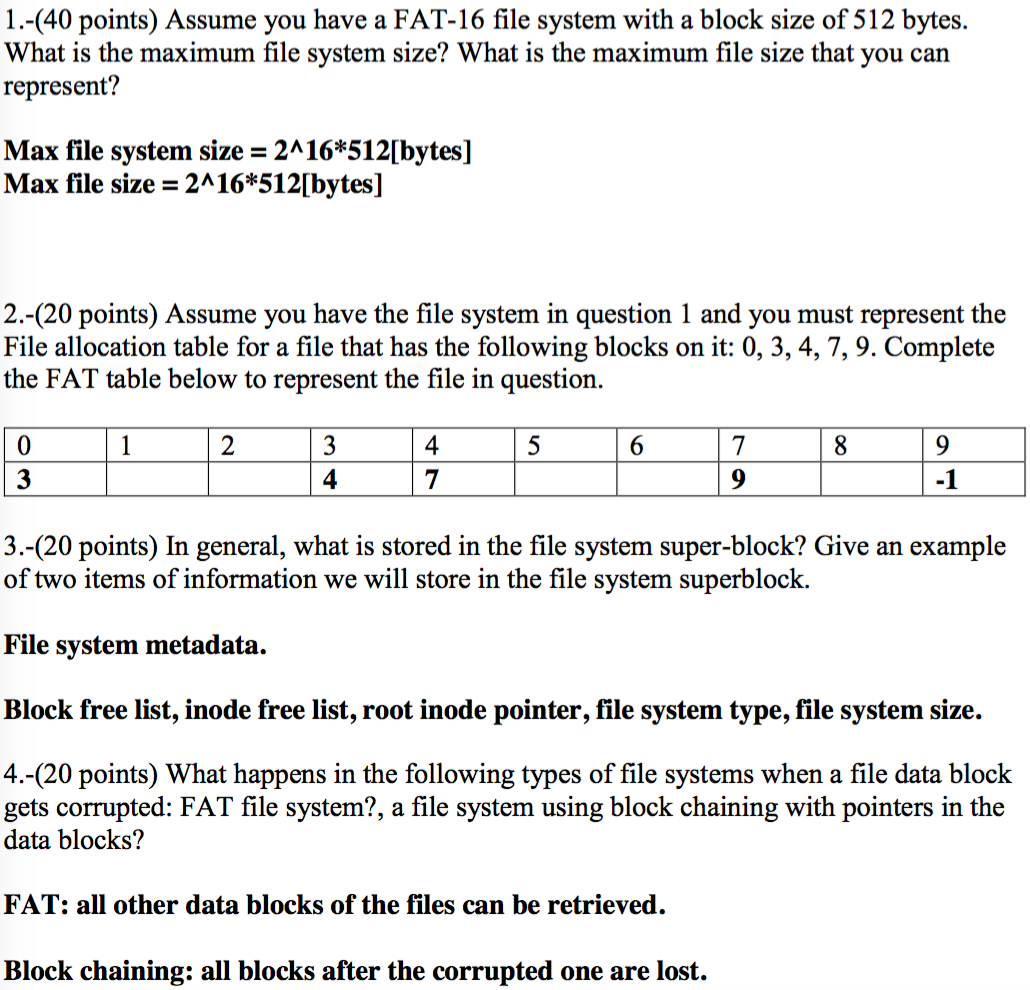


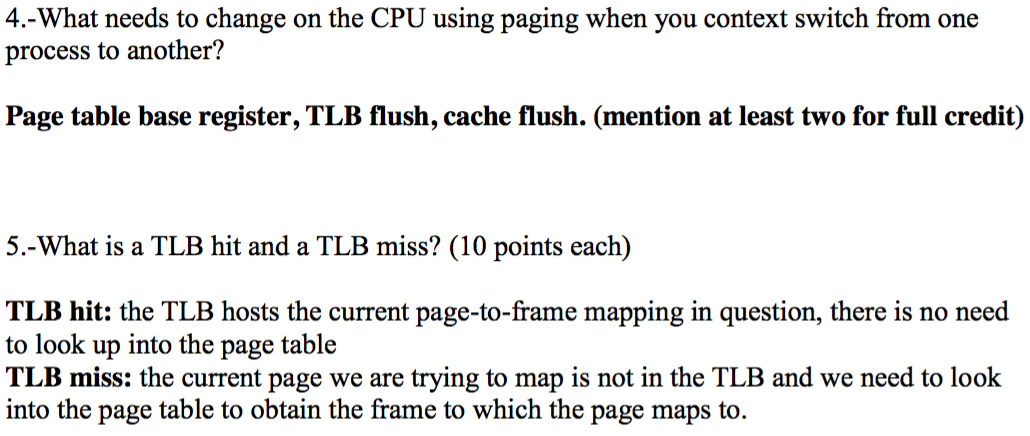


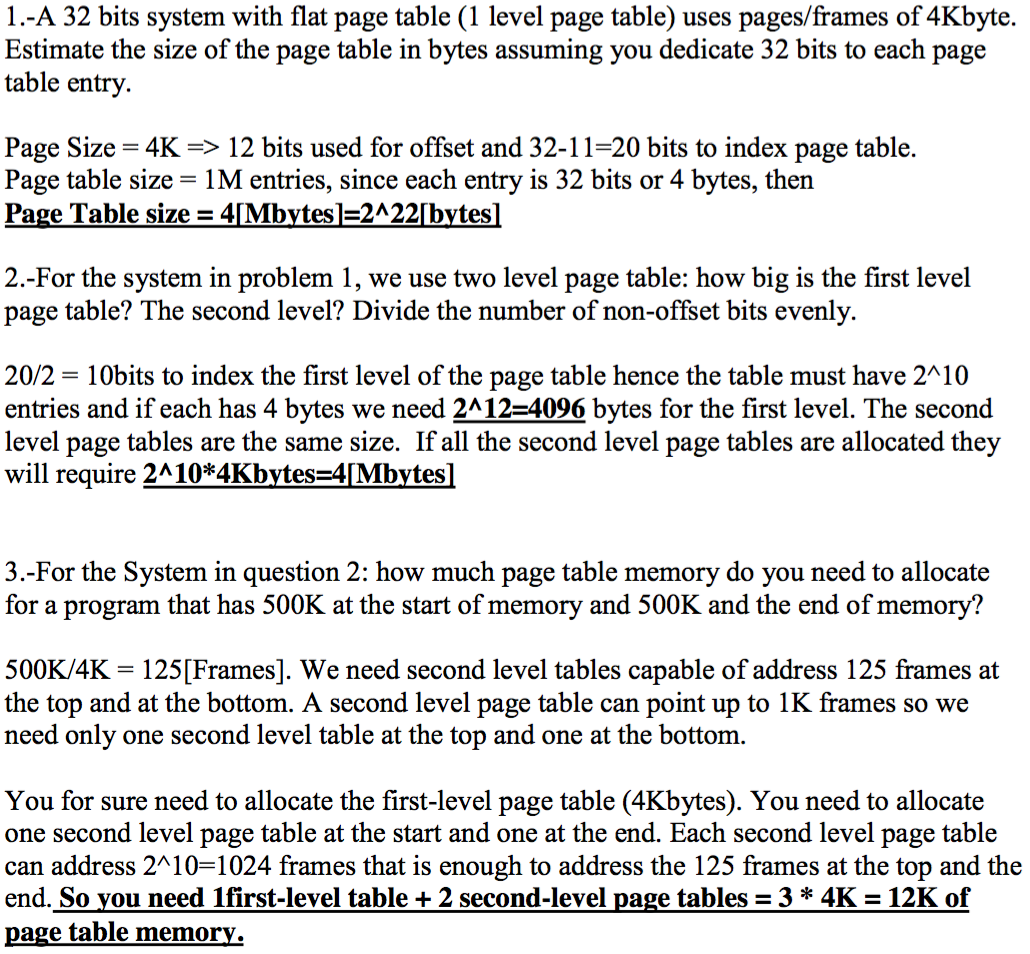


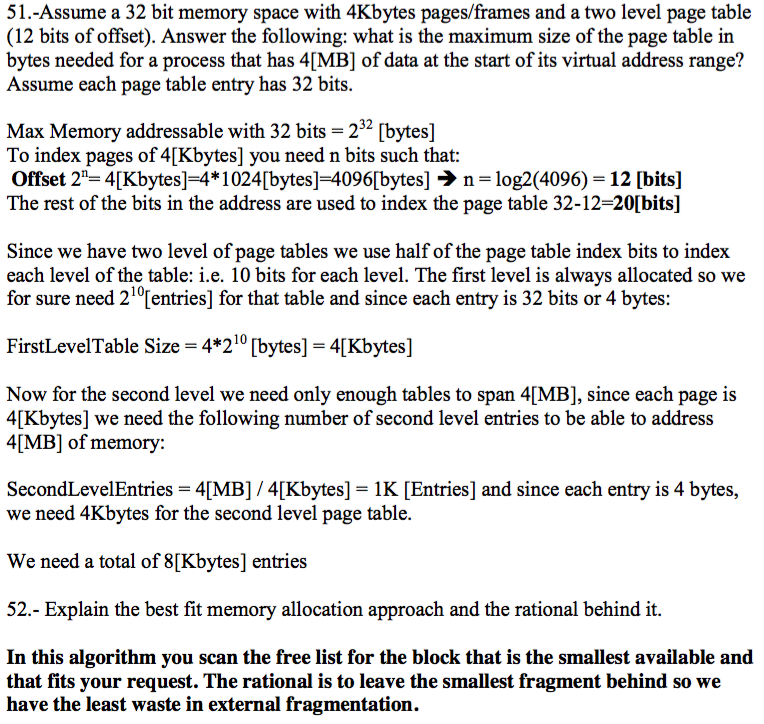












**Macrokernel:** Most OS kernel implemented here. It has better performance but less reliable and secure.

**Microkernel:** Non-essential functionality moved to user-level. It has lower performance but has better usability, portability, and is more secure.

**Layered**: The operating system is divided into layers/levels, each built on top of lower layers. Layer 0 is the hardware and the highest layer is the user interface. With modularity, layers are selected such that each uses fuctions (operations) and services of only lower-level layers.

**Linux Kernel**

Concept: Part of kernel is loaded and compiled on demand when needed, only.

Advantages: Better utilization and only used when needed.

**System Calls**

Fork();// create new process

Kill(); //Send signal to given process

Wait(); //reaps state of child process after termination

Getpid(); //gets process pid

Getppid(); // gets parent pid

A trap is a software interrupt

The interrupt vector is an array of pointers to the interrupt handlers

An interrupt vector is typically set up by the OS

The OS API for the user process is implemented with a trap instruction

POST means Power on Self Test

While running OS code, the process will be in kernel mode

Caching write through works as all writes go to cache only

Computer components are usually connected via a parallel bus

Interrupt handler routines must disable interrupts at the start

On a 32-bit Linux system (3GB memory for process/1GB kernel) you have two processes need topass one chunk of 1GB from one process to the other. Explain whether you will use shared memoryor message passing and explain why your choice will be the best choice.

Since the complete kernel (buffers plus code) is 1GB, there is no way toimplement a buffered IPC that will enable you to pass the chunk of memory you need totransmit. Hence your best choice for this is to use shared memory. This is the most efficientsolution even counting the overhead to set the shared memory from both processes.

-Describe the steps involved in a context switch from one process (P1) to another process (P2)with the level of detail given in class. (10 points each item to a maximum of 50points)

* Operating system saves P1 context in PCB1
* Operating system flush caches used by the processor
* OS invokes the scheduler to fetch the next process to be scheduled, P2 (i.e. PCB2)
* OS loads the state from PCB2 into the processor
* OS points memory context to that of P2
* OS jumps to execute the code of P2

Enumerate the Critical section solutions studied in class and state if they are hardware/software solutions, and whether they have or not busy-wait Interrupt

Disable/Enable: hardware/no busy wait

Peterson’s: software/busy-wait

Swap/TestAndSet: hardware/busy-wait

Locks: software/busy-wait

Semaphores: software/busy-wait or no busy-wait depending on whether blocking or non-blocking semaphore.

Bound-Buffer Problem

N buffers, each can hold one item

Semaphore mutex initialized to 1

Semaphore full initialized to zero

Semaphore empty initialized to the value of n

Do { //producer

//produce item in next\_produced

Wait (empty);

Wait (mutex);

//add next produced to the buffer

Signal (mutex);

Signal (full);

}while (true)

//consumer

Do{

//consumer

wait(full);

wait(mutex);

//removie item from buffer to next\_consumed

signal(mutex);

signal(empty);

//consumer the item in next consumed

}while (true)

FIFO scheduling is when the processes are served in the order they arrived

HOMEWORK 4 THREADS

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <stdbool.h>

#include <signal.h>

#include <sys/wait.h>

#include <sys/types.h>

void \*Thread1 (void \*var);

void \*Thread2 (void \*var);

void signal\_handler(int sig);

pthread\_cond\_t a = PTHREAD\_COND\_INITIALIZER;

pthread\_cond\_t b = PTHREAD\_COND\_INITIALIZER;

pthread\_mutex\_t mut = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_t thread1, thread2;

bool siginter=false;

void \*Thread1 (void \*var){

char \*message;

message = (char \*)var;

int i;

while(siginter==false) {

printf("%s\n", message);

pthread\_mutex\_lock(&mut);

pthread\_cond\_signal(&a);

pthread\_mutex\_unlock(&mut);

pthread\_mutex\_lock(&mut);

while(pthread\_cond\_wait(&b, &mut) != 0){

i++;

}

pthread\_mutex\_unlock(&mut);

printf("thread 1: pong! thread 2 ping received\n");

sleep(3);

}

return NULL;

}

void \*Thread2 (void \*var){

char \*message;

message = (char \*)var;

int i;

while(siginter==false) {

pthread\_mutex\_lock(&mut);

while(pthread\_cond\_wait(&a, &mut) != 0){

i++;

}

pthread\_mutex\_unlock(&mut);

printf("thread 2: pong! thread 1 ping received\n");

printf("%s\n", message);

pthread\_mutex\_lock(&mut);

pthread\_cond\_signal(&b);

pthread\_mutex\_unlock(&mut);

sleep(3);

}

return NULL;

}

void signal\_handler(int sig){

if(sig==SIGINT){

siginter=true;

}

}

int main(void)

{

char \*message1 = "thread 1: ping thread 2";

char \*message2 = "thread 2: ping thread 1";

int t1, t2;

signal(SIGINT, signal\_handler);

t1 = pthread\_create(&thread1, NULL, Thread1, (void \*)message1);

t2 = pthread\_create(&thread2, NULL, Thread2, (void \*)message2);

pthread\_join(thread1,NULL);

pthread\_join(thread2,NULL);

if(siginter==true) {

exit(0);

}

return 0;

}

HOMEWORK 3 PIPES

int main(void){

int fd[2];

int pk[2];

pid\_t pid;

if (pipe(fd) == -1){

puts("Pipe fd failed :(");

return 1;

} //create pipe

if (pipe(pk) == -1){

puts("Pipe pk failed :(");

}pid = fork();

char sendone[50];

char sendtwo[50];

snprintf(sendone, 50, "I am your Daddy! and my name is %d \n", (int)getppid());

snprintf(sendtwo, 50, "Daddy, my name is %d \n", (int)getpid());

if (pid>0){

close(fd[0]);

write(fd[1], sendone, strlen(sendone)+1);close(fd[1]);

close(pk[1]);read(pk[0], sendtwo, 50);

printf("%s", sendtwo);close(pk[0]);

}

else{

close(fd[1]);read(fd[0], sendone, 50);printf("%s", sendone);

close(fd[0]);

close(pk[0]);

write(pk[1], sendtwo, strlen(sendtwo)+1);

close(pk[1]);

}

return 0;

}

HOMEWORK2

#include<stdio.h>

#include<signal.h>

#include<unistd.h>

#include<stdlib.h>

#include<stdbool.h>

bool sigcheck = false;

void sig\_handler(int signo)

{ if (signo == SIGINT){

if (sigcheck==true) {

sigcheck = false;

}

else {

sigcheck=true;

}

}

else if (signo == SIGUSR1) {

printf("\nreceived SIGUSR1\n");

exit (0);

}

}

int main(void)

{

int count =0;

while(1) {

signal(SIGUSR1, sig\_handler);

if (sigcheck == true) {

printf("\nIteration: %i\n", count);

}

signal(SIGINT, sig\_handler);

count++;

sleep(2);

}

return 0;

}