Chapter 1

Operating System: exploits the hardware resources of one or more processors. Provided a set of services to system users. Manages secondary memory and I/O devices.
Microprocessor: Invention that brought about desktop and handheld computing. Processor on a single chip. Fastest general-purpose processor. Multiprocessors. Each chip (socket) contains multiple processors (cores).
Main memory: Referred to as real memory or primary memory.

IR - PC = Opcode
Mem range = PC
Data range = IR
Number of possible opcodes = 2^Opcode
Memory size = 2^PC * IR
Memory range = PC
Data Range = IR/PC

AAT = HR(T1) + MR(T1+T2)
Where, MR = 1-HR

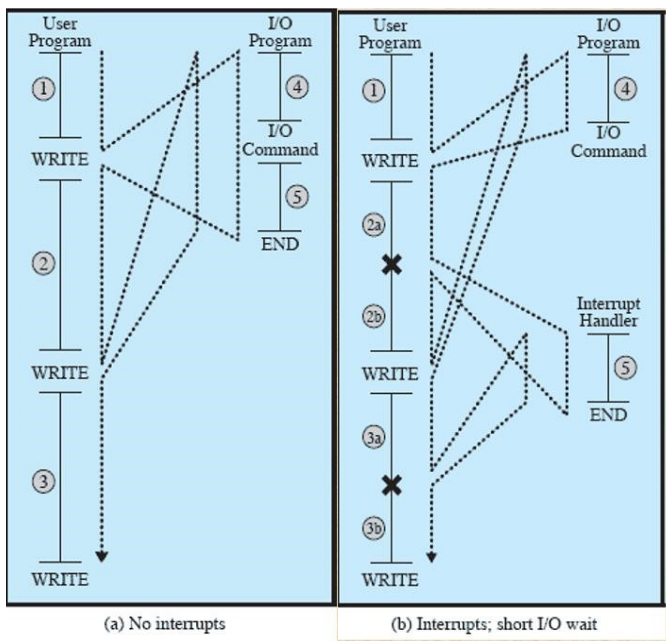
HEX notation
Mem range = 000 - FFF
Data range = 0000 - FFFF

Octal notation
Mem range = 000 - 777
Data range = 0000 - 777

Interrupts: Interrupt the normal sequencing of the processor. Provided to improve processor utilization.

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Multiple Interrupts: Two approcahes -> disable interrupts while an interrupt is being a processed ->use a priority scheme. (Sequential or Nested.)
Memory Hierarchy: constraints (amount, speed, expense). Must be amle to keep up with processor. Cost of memory must be reasonable in a relationship to the other components.
Principle of Locality: memory references by the processor tend to cluser. Data is organized so that percentage of accesses to each successively lower level is substantially less than that of the level above. Can be aplied across more than two levels of memory.
Cache memory: Interacts with other memory management hardware. Processor must access memory at least once per instruction cycle. Processor execution is limited by memory cycle time. Exploit the principle of locality with a small fast memory.
Cache principles: Contains a copy of a portion of main memory. Processor first checks cache. If not found, a block of memory is read into cache. Because of locality of reference, it is likely that many of the future memory references will be to other bytes in the block. Mapping Function: Determines which cache location the block will occupy.
LRU – Effective strategy is to replace a block that has been in the cache the longest with no references to it. Hardware mechanisms are needed to identify the least recently used block. Chooses which block to replace when a new block is to be loaded into the cache.
I/O Techniques: Programmed I/O, Interrupt Driven I/O, Direct Memory Access (DMA) Programmed I/O: I/O module performs the requested action then sets the appropriate bits in the I/O status register. Processor periodically checks the status of the I/O module until it determines the instruction is complete. With programmed I/O the performance level of the entire system is severely degraded.
Interrupt Driven I/O drawbacks: Transfer rate is limited by the speed with which the processor can test and service a device. The processor is tied up in managing an I/O transfer.
Direct memory access: transfer the entire block of data directly to and from memory without going through the processor. Processor is involved only at the beginning and end of the transfer. Processor executes more slowly during a transfer when processor access to the bus is required. More efficient than interrupt-driven or programmed I/O.
Symmetric Multiprocessors: Two or more similar processors of comparable capability. Processors share the same main memory and are interconnected by a bus or other interna; connection scheme. Processors share access to I/O devices. All processors can perform the same functions. The system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels
SMP Advantages: Performance (a system with multiple processors will yield greater performance if work can be done in parallel.) Scaling (vendors can offer a range of products with different price and performance characteristics.) Availability (the failure of a single processor does not halt the machine.) Incremental growth (an additional processor can be added to enhance performance)



(a) No interrupts (b) Interrupts; short I/O wait

Average access time = Hit Ratio * TL1 + Miss ratio * (TL2+TL1)
#blocks in main memory = 2^n / K

Chapter 2

Key Interfaces – ISA, ABI, API
Role of OS – A computer is a set of resources for the movement, storage, and processing of data. The OS is responsible for managing these resources.
OS a software – Functions in the same way as ordinary computer software. Program, or suite of programs, executed by the processor. Frequently relinquishes control and must depend on the processor to allow it to regain control.
Serial Processing – Earliest Computers (No OS. Programmers interacted directly with the computer hardware. Computers ran from a console with display lights, toggle, switches, some form of input device and a printer. Users have access to the computers in “series”. Problems (Scheduling: most installations used a hard copy sign-up sheet to reserve computer time. Time allocations could run short or long, resulting in wasted computer time. Set up time: a considerable amount of time was spent just on setting up the program to run.
Simple Batch Systems

- Monitor Point of View - Monitor controls the sequence of events. *Resident Monitor* is software always in memory. Monitor reads in job and gives control. Job returns control to monitor
- Processor Point of View - Processor executes instruction from the memory containing the monitor. Executes the instructions in the user program until it encounters an ending or error condition. “*control is passed to a job*” means processor is fetching and executing instructions in a user program. “*control is returned to the monitor*” means that the processor is fetching and executing instructions from the monitor program Modes of Operation:
- User mode: User program executes in user mode. Certain areas of memory are protected from user access. Certain instructions may not be executed
- Kernel Mode: Monitor executes in kernel mode. Privileged instructions may be executed. Protected areas of memory may be accessed

Simple Batch System Overhead

- Processor time alternates between execution of user programs and execution of the monitor
- Sacrifices:
  o some main memory is now given over to the monitor o some processor time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

Uniprogramming - The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding. Multiprogramming - There must be enough memory to hold the OS (resident monitor) and one user program. When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O

Time Sharing System - Can be used to handle multiple interactive jobs. Processor time is shared among multiple users. Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation.
Development of the process

- Multiprogramming batch operation: processor is switched among the various programs residing in main memory.
- Time Sharing: be responsive to the individual user but be able to support many users simultaneously
- Real-time transaction systems: a number of users are entering queries or updates against a database Causes of errors:
- Improper Synchronization: a program must wait until the data are available in a buffer. Improper design of the signaling mechanism can result in loss or duplication.
- Failed Mutual Exclusion: more than one user or program attempts to make use of a shared resource at the same time. Only one routine at time allowed to perform an update against the file.
- Nondeterminate program operation: program execution is interleaved by the processor when memory is shared. The order in which programs are scheduled may affect their outcome
- Deadlocks: it is possible for two or more programs to be hung up waiting for each other. May depend on the chance timing of resource allocation and release.

Process Management: The entire state of the process at any instant is contained in its context. New features can be designed and incorporated into the OS by expanding the context to include any new information needed to support the feature.
Memory Management: The OS have five principal storage management responsibilities: process isolation, automatic allocation and management, support of modular programming, protection and access control, long-term storage.
Virtual memory - A facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available. Conceived to meet the requirement of having multiple user jobs reside in main memory concurrently
Paging: Allows processes to be comprised of a number of fixed-size blocks, called pages. Program references a word by means of a virtual address. Provides for a dynamic mapping between the virtual address used in the program and a real (or physical) address in main memory.
Information Protection and Security: The nature of the threat that concerns an organization will vary greatly depending on the circumstances. The problem involves controlling access to computer systems and the information stored in them.
Multithreading: Technique in which a process, executing an application, is divided into threads that can run concurrently. Thread: dispatchable unit of work. Includes a processor context and its own data area to enable subroutine branching. Executes sequentially and is interruptible. Process: a collection of one or more threads and associated system resources. Programmer has greater control over the modularity of the application and the timing of application related events.
Symmetric Multiprocessing: Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture. Several processes can run in parallel. Multiple processors are transparent to the user. The OS takes care of scheduling of threads or processes on individual processors and of synchronization among processors.

Chapter 3

OS Management of Application Execution: Resources are made available to multiple applications. The processor is switched among multiple applications so all will appear to be progressing. The Processor and I/O devices can be used efficiently.
Two essential elements of a process are: Program code ->which may be shared with other processes that are executing the same program. A set of data associated with that code -> when the processor begins to execute the program code, we refer to this executing entity as a process.
Process Elements: identifier, state, priority, program counter, memory pointers, context data, I/O status information, accounting information.
Process states: Trace -> the behavior of an individual process by listing the sequence of instructions that execute for that process -> the behavior of the processor can be characterized by showing how the traces of the various processes are interleaved. Dispatcher -> small program that switches the processor from one process to another.
Process creation: Process spawning -> when the OS creates a process at the explicit request of another process. Parent process -> is the original process. Child process -> is the new process. Process Termination -> There must be a means for a process to indicate its completion. -> A batch job should include a HALT instruction or an explicit OS service call for termination. -> For an interactive application, the action of the user will indicate when the process is completed (e.g., log off, quitting an application)
Swapping -> involves moving part of all a process from main memory to disk. -> when none of the processes in main memory is in the Ready state, the OS swaps one of the blocked processes out on to disk into a suspend queue.
Characteristics of a suspended process -> The process is not immediately available for execution -> The process may or may not be waiting on an event -> The process was placed in a suspended state by an agent: either itself, a parent process, or the OS, for the purpose of preventing its execution -> The process may not be removed from this state until the agent explicitly orders the removal.
Memory Tables -> Used to keep track of both main (real) and secondary (virtual) memory -> Processes are maintained on secondary memory using some sort of virtual memory or simple swapping mechanism.

- Must include: allocation of main memory to processes, allocation of secondary memory to processes, protection attribute of blocks of main or virtual memory. Information needed to manage virtual memory.

I/O Tables -> Used by the OS to manage the I/O devices and channels of the computer system -> At any given time, an I/O device may be available or assigned to a particular process.

- These tables provide information about: existence of files, location on secondary memory, current status, other attributes.

Process tables -> must be maintained to manage processes -> There must be some reference to memory I/O, and files, directly or indirectly -> The tables themselves must be accessible by the OS and therefore are subject to memory management.
Process Control Structures -> OS must know where the process is located -> the attribute of the process that are necessary for its management.
Process Location -> A process must include a program or set of programs to be executed -> A process will consist of at least sufficient memory to hold the programs and data of that process ->The execution of a program typically involves a stack that is used to keep track of procedure calls and parameter passing between procedures
Process Attributes -> Each process has associated with it a number of attributes that are used by the OS for process control -> The collection of program, data, stack, and attributes is referred to as the process image -> Process image location will depend on the memory management scheme being used
Process Identification -> Each process is assigned a unique numeric identifier -> Many of the tables controlled by the OS may use process identifiers to cross-reference process tables -> Memory tables may be organized to provide a map of main memory with an indication of which process is assigned to each region -> When processes communicate with one another, the process identifier informs the OS of the destination of a particular communication -> When processes are allowed to create other processes, identifiers indicate the parent and descendants of each process.
Processor State Information: Consists of the contents of processor registers -> user-visible registers -> control and status registers -> stack pointers. Program status word (PSW) -> contains condition codes plus other status information -> EFFLAGS register is an example of a PSW used by any OS running on an x86 processor.
Process Control Information: The additional information needed by the OS to control and coordinate the various active processes
Role of the process control block -> The most important data structure in an OS -> contains all of the information about a process that is needed by the OS -> blocks are read and/or modified by virtually every module in the OS -> defines the state of the OS. Difficulty is not access, but protection -> a bug in a single routine could damage process control blocks, which could destroy the system’s ability to manage the affected processes -> a design change in the structure or semantics of the process control block could affect a number of modules in the OS.
Modes of Execution:

• User Mode -> less-privileged mode -> user programs typically execute on this mode • System Mode -> more-privileged mode -> also referred to as control mode or kernel mode -> kernel of the operating system.

Process creation (step by step): assigns a unique process identifier to the new process, allocates space for the process, initializes the process control block, sets the appropriate linkages, creates or expands other data structures.
System Interrupts: Interrupt -> Due to some sort of event that is external and independent of the currently running process (clock interrupt, I/O interrupt, memory interrupt) -> Time slice (the maximum amount of time that a process can execute before being interrupted). Trap -> An error or exception condition generated within the currently running process -> OS determine if the condition is fatal. If no interrupts are pending the processor -> proceeds to the fetch stage and fetches the next instruction of the current program in the current process.
If an interrupt is pending the processor -> sets the program counter to the starting address of an interrupt handler program -> switches from user mode to kernel mode so that the interrupt processing code may include privileged instructions
Change of process state (step by step): save the context of the processor, update the process control block of the process currently in the running state, move the process control block of this process to the appropriate queue, select another process for execution, update the process control block of the process selected, update memory management data structures, restore the context of the processor to that which existed at the time the selected process was last switched out.
Security issues -> An OS associates a set of privileges with each process -> Typically a process that executes on behalf of a user has the privileges that the OS recognizes for that user -> Highest level of privilege is referred to as administrator, supervisor, or root access -> A key security issue in the design of any OS is to prevent, or at least detect, attempts by a user or a malware from gaining unauthorized privileges on the system and from gaining root access
System access threats: Intruders -> often referred to as a hacker or cracker -> objective is to gain access to a system or to increase the range of privileges accessible on a system -> attempts to acquire information that should have been protected. Malicious software -> most sophisticated types of threats to computer system -> can be relatively harmless or very damaging
Countermeasures access control: Implements a security policy that specifies who or what may have access to each specific system resource and the type of access that is permitted in each instance -> Mediates between a user and system resources -> A security administrator maintains an authorization database -> An auditing function monitors and keeps a record of user accesses to system resources Countermeasures Firewall: A dedicated computer that -> interfaces with computers outside a network -> has special security precautions built into it to protect sensitive files on computers within the network. Design goals of a firewall -> all traffic must pass through the firewall -> only authorized traffic will be allowed to pass -> immune to penetration

Chapter 4

Processes 2 characteristics

- Resource Ownership: process includes a virtual address space to hold the process image.
- Scheduling/Execution: Follows an execution path that may be interleaved with other processes.

Processes and threads -> The unit of dispatching is referred to as a thread or lightweight process -> The unit of resource ownership is referred to as a process or task -> Multithreading - The ability of an OS to support multiple, concurrent paths of execution within a single process
Single Threaded Approach: A single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach

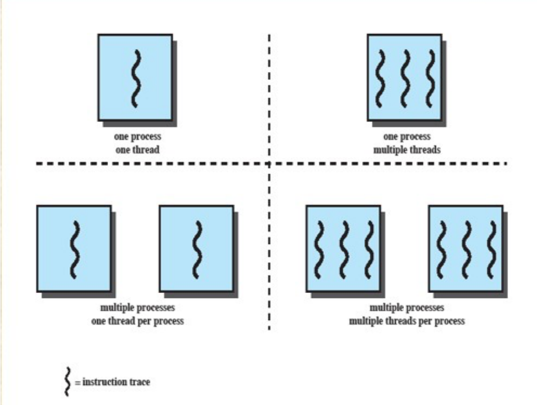

one process one thread
one process multiple threads
multiple processes one thread per process
multiple processes multiple threads per process
= instruction trace

Figure 4.1 Threads and Processes [ANDE97]

Processes:

- The unit or resource allocation and a unit of protection
- A virtual address space that holds the process image
- Protected access to: o Processors o Other processes o Files o I/O resources

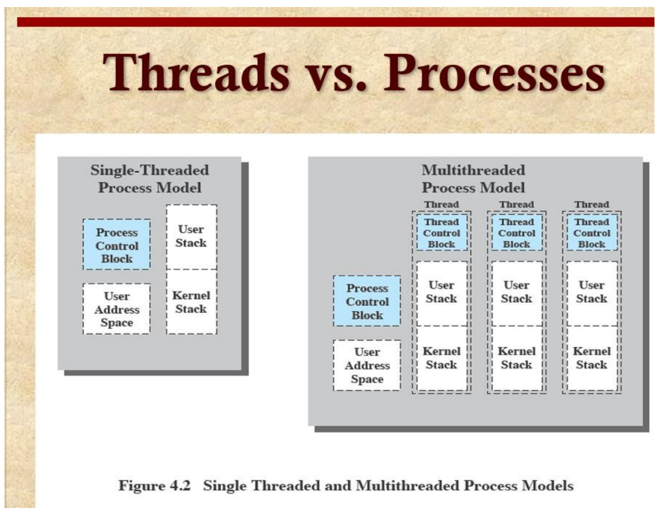


Figure 4.2 Single Threaded and Multithreaded Process Models

One or more threads in a process: Each thread has -> an execution -> saved thread context when not running -> an execution stack -> some per-thread static storage for local variables -> access to the memory and resources of its process (all threads of a process share this)
Benefits of Threads -> Takes less time to create a new thread than a process -> less time to terminate a thread than a process -> switching between two threads takes less time than switching between processes -> Threads enhance efficiency in communication between programs
Thread use in a single-user system -> Foreground and background work -> Asynchronous processing -> Speed of execution -> Modular program structure
Suspending Threads -> suspending a process involves suspending all threads of the process -> termination of a process terminates all threads within the process
Thread execution states: Key states (Running, ready, blocked), Thread operations associated with a change in thread state are (Spawn, block, unblock, finish)
Thread Synchronization: It is necessary to synchronize the activities of the various threads -> all threads of a process share the same address space and other resources -> any alteration of a
resource by one thread affects the other threads in
the same process Types of threads: User level
thread (ULT) and Kernel level thread (KLT)
User-level threads -> all thread management is done by the application -> the kernel is not aware of the existence of threads
Advantages of ULTs: Thread switching does not require kernel mode privileges -> scheduling can be application specific -> ULTs can run on any OS
Disadvantages of ULTs: In a typical OS many system calls are blocking, as a result, when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked -> In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing
Overcoming ULT disadvantages: Jacketing -> converts a blocking system call into a non-blocking system call -> writes an application as multiple process rather than multiple threads. Kernel-Level threads (KLTs) -> thread management is done by kernel -> no thread management is done by the application -> Windows is an example of this approach
Advantages of KLTs -> The Kernel can simultaneously schedule multiple threads from the same process on multiple processors -> If one thread in a process is blocked, the kernel can schedule another thread of the same process -> Kernel routines can be multithreaded
Disadvantages of KLTs: The transfer of control from one thread to another within the same process requires a mode switch to the kernel
Combine approaches -> thread creation is done in the user space -> bulk of scheduling and synchronization of threads is by the application -> Solaris is an example

Libraries: pthread.h, stdio.h, stdlib.h, sys/types.h, sys/wait.h
Declaration: pid_t id;
Threads Syntax: if(pthread_create(&id[i], NULL, name_of_function,&dynamic_or_static_arrName[i]))

for(int i = 0; i < size; i++) pthread_join(id[i], NULL)

struct struct_name* your_name = (struct* struct_name) name_of_param;

```
void *calculator(void *pos_void_ptr) {


struct operation *pos_ptr = (struct operation *)pos_void_ptr;


//cast pos_void_ptr to a struct operation *


switch(pos_ptr->op) {


[...] }


int main() {


static struct operation operations[NOPER]; pthread_t tid[NOPER];


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


operations[i].op = i;
std::cin >> operations[i].val1;
std::cin >> operations[i].val2;
if(pthread_create(&tid[i], NULL, calculator, &operations[i]))


//Call pthread_create


{
[...]


} }


// Wait for the other threads to finish.


for (int i = 0; i < NOPER; i++)
pthread_join(tid[i], NULL); //Call pthread_join here


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


switch(operations[i].op) {


[...] }


}
```

```
int main() {


pid_t pid;
std::cout << "I am the parent process" << std::endl; for(int i=0;i<3;i++)
{


pid = fork(); if (pid == 0) {


std::cout << "I am the child process " << i << std::endl; if (i==1)
{


pid = fork(); if (pid == 0) {


std::cout << "I am a grandchild process from child process " << i <<
std::endl;


_exit(0); }


wait(0); }


_exit(0); }


wait(0); }


return 0; }
```

Extras:

What is the major disadvantage of not having privileged instructions? A. Unrestricted user access to the data on the hard drive. **In a vectorized interrupt system, an interrupt can only be interrupted by an interrupt of higher priority**. True. **A program in execution is called.** A process. **What is the major reason for the success of modular kernels?** They let users add extensions to the kernel. **Each process has its own Process Control. Block.** True. **Which of these events can move a process from the running state to the blocked state?** The process performs a system call. **Which of the following statements apply to the program.cs.uh.edu server?** It is an interactive & time-sharing system. **Which one of the following is not shared by threads that share the same address space?** Their stacks & program counters. **Which system call returns the process ID of a terminated child?** Wait(). **The time required to create a new thread in an existing process is:** Less than the time required to create a new process. **The execv() system call specifies which new program a process should execute.** True. **The execv() system call creates a new process.** False. **Which hardware mechanism allows a device to notify the CPU of an event?** A. Interrupts. **Which of these events can move a process from the running state to the ready state?** A timer interrupt. The arrival in the ready state of a higher priority process. **Which of these events can move a process from the running state to the blocked state?** The process performs a system call. **In which queue is a newly created process initially put?** Ready queue**. Memory protection is normally done through privileged instructions.** False. **Delaying disk— or SSD—writes.** May result in lost data if the system crashes. **Which of the following statements does not apply to microkernels?** They are faster than most other kernel organizations. **A process in the ready state can only move from that state to the:** Running State. **Which of the following actions are the normal result of a system call?** An interrupt occurs**. What is the default action a Linux process takes when it receives a signal from another process?** It terminates.

The goal of the suspend state in a five-state process model with one suspend state is to making room in main memory for new process by moving the process in the ready state to virtual memory. False. **The main objective of a time-sharing system is to reduce the response time?** True. **In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing?** True. **Does a small cache block size improve the hit ratio of the principal of locality.** False. **Does a system call generate a change of mode (from KERNAL MODE TO USER MODE)**. False. **In an interrupt-driven i/o call, the processor is the resource that handles the transfer of information between the I/o devices and memory?** True. **Select the thread implementation where threads must use the scheduler provided by the OS.** Kernel-level threads. **Select the process state where process reside in main memory and can be chosen by the scheduler to be executed.** Running. **The kernel structure that includes virtually all the OS functionalities is a.** Monolithic Kernel. **What type of interrupt is a division by zero.** Program. **Select the multiple interrupt handling mechanism that disables interrupts while an interrupt is being processed.** Multi-Interrupts; Sequential. **The process that executes the instruction after if((pid=fork())==0).** Child Process.

PID - An element of the process control block.
pthread library - A user-level thread implementation.
Multiple Interrupts: Sequential - Disable interrupts while an interrupt is being processed.
if ((pid=fork()) == 0) - Child process code.
a.*What is the goal of the suspend state in the five-state process modelwithone suspendstate?*
To free a space in main memory and bring in another process b.*What is the main objective of a time-sharing system?*
Minimize response time of processes
c.*Describe why in a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing.*
The single-threaded process at the kernel level that represents the multithreaded processes at the user level canonlyruninoneCPUata particulartime
d.*What is the relationship between the cache memory and the block size?* **Big block size = small number of records in the cache memory (replacement probability increases.**
Small block size = destroying the principle of locality.
e.*What is the major disadvantage of a system call?*
Change from user mode to supervised mode = context switch. f.*What is the major advantage of DMA against interrupt-driven I/O?*
DMA is performed by a separate module on the system bus or incorporated into an I/O module, therefore the processor is notinterruptedduringword transfer.TheCPUisonlyusedat
the beginning and end of the transfer.

Additional notes:

Chapter 1:
- Priority based interrupt handler does not disable an interrupt which is being processed to execute new interrupt.
- Memory must be able to keep up with processor.
- We cannot access blocks stored in cache memory.
- Processor first checks cache, if not found brings block from memory to cache.
- LRU algorithm is used to replace block in the cache (longest with no references)
- Memory write operation should take place when block is replaced in cache.(preferred)

- Symmetric Multiprocessors(SMP): 2 or more processors with comparable capacity, share same main memory and are interconnected by a bus or other connection scheme, share I/O devices, perform same functions
- Multicore/ multichip: combine two or more processors on a single chip, have L2 or sometimes L3 cache.
- IR - PC = Opcode
- Mem range = PC
- Data range = IR
- Number of possible opcodes = 2^Opcode
- Memory size = 2^PC * IR
- Memory range = PC
- Data Range = IR/PC
- AAT = HR(T1) + MR(T1+T2)
- Where MR = 1-HR

Chapter2:
- Kernel: functions of OS that are always in memory.
- OS relies on processor to regain control that it frequently relinquishes.
- “control is passed to a job” means processor is fetching and executing instructions in a user program
- “control returned to the monitor” means that the processor is fetching and executing instructions from the monitor
- User mode: user program executes in user mode, certain areas of memory protected from user access, certain instructions may not be executed
- Kernel mode: monitor executes in this mode, privileged instructions may not be executed, protected areas of memory may be accessed.
- Compared to uniprogramming. Multiprogramming reduces the elapsed time and throughput but processor, disk and memory use increase
- Process: contains executable program, associated data needed by the program, and execution context
- Execution context: is essential: internal data by which OS is able to supervise and control the process, includes the contents of various registers, and info such as priority of process and whether the process is waiting for the completion of a particular I/O event
- Virtual memory: to meet requirement of having multiple user jobs reside in main memory concurrently, allows programs to address memory from a logical point of view, without regard to amount of main mem physically available
- Paging: allows processes to be comprised of number of fixed sized blocks, called pages. Program references word by means of a virtual address. Page may be located anywhere in memory. Provides dynamic mapping betn. Virtual addr. Used in the program and real address in main mem.
- SMP Advantages: more than one process can be running simultaneously in different processors, single process halt does not affect the whole system, performance can be enhanced by adding processor, easier to scale because different products can be offered depending on no. of processors available in the system
- Distributed OS: provides illusion of single memory space, single secondary memory space, and unified access facilities. State of art for distributed OS lags that of uniprocessor and SMP operating systems
- Object Oriented Design: for adding modular extensions to small kernel, enables programmers to customize OS without disrupting system intregrity. Eases development of distributed tools
- Virtual machines: single PC or server to simultaneously run multiple OS or multiple sessions of a single OS. Can host numerous apps. Host OS can support number of virtual machines

Chapter 3 & 4:
- Process Control Block: Identifier, State, priority, program counter, memory pointers, context data, I/O status info, accounting info
- Process: program in execution
- Thread: unit of execution within a process. Single process can have one to many threads
- Process State: trace and dispatch.
- Program counter: address of next line of instruction that needs to be executed
- Context switch: basically bookmark for a process that got interrupted while it was being executed. Saved in PCB
- Process spawning: OS creates new process at the explicit request of another process
- Process creation reason: new batch job, interactive logon (user in terminal), by OS to provide service, spawned by existing process
- Process termination reason: completion, time limit, memory, bounds violation, protection err., arithmetic err., time overrun, I/O failure, invalid instr, privileged instr, data misuse, parent request/termination, OS intervention
- Process suspension reason: swapping, OS reason, user request, timing, parent process request
- Suspended process cannot be executed unless it is moved to ready state
- Process image: user data, user program, stack, PCB
- Process attributes: identifier, user-visible registers, control and status registers, stack pointers
- Program Status Word(PSW): contains condition codes plus other status information , EFLAGS register is an example of a PSW used by any OS running on an x86 processor

Memory range 0-7 for octal, adding another 0 and 7 for every octal digit in PC.
(So if the info given says Pc: 3 octal digits it's 000-777)
For Hex it's 0-F with the same rule as above.
(So for PC: 2 Hex digits its 00-FF)
Data range for octal is IR (in bits)/(3)
Data range for hex is IR (in bits)/(4)

So for the practice we had 12 bits in octal notation so its: 12/3 = 4 So Data range was: 0000-7777

2^ (IR - PC) = # OPCode

```
struct arg
{
   string value;
   std::string *array;
};

string helperFunction(string value){
   return value+"+add";
}

void *inc_x(void *x_void_ptr)
{
   struct arg *x_ptr = (struct arg *)x_void_ptr;          //this
   *(x_ptr->array) = helperFunction((x_ptr->value));  //this
   return NULL;
}

int main()
{
   static struct arg x[5];
   pthread_t tid[NTHREADS];
   std::string result[5];
   std::string stringname = "Hello";

   for(int i=0;i<NTHREADS;i++)
   {
      x[i].value = stringname[i];                        //this
      x[i].array = &result[i];                         //this
      if(pthread_create(&tid[i], NULL, inc_x, &x[i]))     //this
      {
         fprintf(stderr, "Error creating thread\n");
         return 1;
      }
   }

   // Wait for the other threads to finish.
   for (int i = 0; i < NTHREADS; i++)            //this
         pthread_join(tid[i], NULL);

   for (int i = 0; i < NTHREADS; i++)
      std::cout << result[i] <<" ";
   return 0;
}
```