**C How to Program Notes-**

Pointers- Example: int count = 7;

Int \*countPtr = &count;

Dereferencing a pointer- Example: printf(“%d”, \*yPtr); using it in this manner is dereferencing.

How to make C programs that satisfy the following two conditions? Efficient by writing functions using call-by-reference. Keeping the values of the function arguments not modified during function calls.

Using the const qualifier with pointers. The const qualifier informs the compiler that the value of the particular variable should not be modified.

A pointer to a function contains the address of the function in memory.

Why do we need function malloc()?- malloc() is used to allocate a certain amount of memory during the execution of a program.

**Ch 1- Introduction to OS**

With the user mode, can we execute some system functions of

the OS?

What are the two main functions of the OS?

Operating Systems provide user programs (and app programmers) a clean abstract set of resources instead of messy hardware ones. It also manages all these hardware resources.

How do the CPU, memory, and I/O devices exchange data and communicate with one another?

A system bus is a single computer bus that connects the major components of a computer system, combining the functions of a data bus to carry info, an address bus to determine where it should be sent, and a control bus to determine its operation.

What does a CPU do?

Fetches the first instruction from memory, decodes and executes it. Then fetch, decode and execute subsequent instructions.

CPU design-

Traditional- only one instruction at a time

Modern- more than one instruction at a time. Pipeline and superscalar.

Pipeline design- a CPU has a separate fetch, decode and execute units. E.g., executing instructions n, decoding n+1, and fetching n+2 at the same time.

Superscalar design- multiple execution units are present, 2 or more instructions are fetched at once, decoded and dumped into a buffer until they can be executed. As soon as an exe unit available, OS looks the buffer to see any instruction can be executed. An issue that can occur with this is complexity and time cost of dependency checking logic and register renaming circuitry.

When to put a new item into the cache?

e.g., every cache miss for caching lines of main memory

which cache line to be replaced by a new item?

e.g., computed by using some of the high-order bits of the memory addresses referenced

**Chapter 2 Processes and Threads**

It is not recommended to invoke the system call fork() in a for-loop structure to create multiple processes in a C program because each process takes up memory on the CPU.

Why would outputs be out of order?

Printf() vs write()- printf() is buffered and writef() is not. Printf() wont get the output immediately and so outputs may be out of order, writef() fixes this issue.

What is a thread?

A kind of lighted-weighted process within a process, an execution of a process.

Example of multi-threaded program?

A word processor is a multi-threaded program that typically contains 3 threads, user interaction, page formatting, and automatic saving.

Advantages of multi-threaded programming-

System resources can be saved, threads are lighter weight than processes, it is faster and easier to create and dispose of than processes, can speed up apps (more efficient)

Disadvantages of multi-threaded programming-

Synchronization overhead for consistency of shared data, shared process memory space, and program debugging is challenging.

Threads vs Processes

Processes are independent, each has its own address space. All threads belonging to the same process share the address space of the process.

When to choose threads and when to choose processes for multi-task programming?

Threads are useful when you have a workload consisting of light-weighted tasks(in terms of cpu processing time or memory), ex word processor

Processes are useful for parallel programming with several workloads where task take significant computing power, memory or both. Ex. Printing a word or pdf file can sometimes take significant amounts of time and significant memory and I/O operations.

Pthread\_join() function waits for the thread specified by thread to terminate, if that thread is already terminated, then pthread\_join() returns immediately. The function prototype of pthread\_join() is int pthread\_join(pthread\_t threadID, void \*\* retval);

•Typical usage

pthread\_join

(&theadID, NULL);

**Ch 2. IPC**

What are the issues related to Interprocess Communication?

Issues related to IPC-

Process synchronization issues

The order of execution is critical when dependencies of processes are present

Race Condition- two or more processes that are working together share some common storage that each one can read and write. The shared storage may be a shared memory location, or a shared file stored in a hard drive. A situation like this is called a race condition if the final result depends on which process runs first.

Critical sections- is the part of a program (several lines of code) where the shared memory is accessed.

How to avoid race conditions?

Need to find a way to prohibit more than one process from reading and writing shared memory at the same time. Mutual exclusion for processes is necessary.

When an OS is designed, race conditions should be avoided for processes and threads. What are the requirements to avoid race conditions?

What is the problem with strict alternation implementation using busy wait?

Requires that two processes strictly alternate in executing their CRs.

Does it violate any requirements of race conditions?

It violates the 3rd condition to avoid race conditions.

Requirements to avoid: no two processes may be simultaneously inside their critical sections, no assumptions may be made about relative speeds of processes or number of CPUs, no process running outside its critical section may block other processes (efficiency) and no process should have to wait forever to enter its critical section (fairness).

Peterson’s Solution- the first software solution to the mutual exclusion problem that does not require strict alteration, combines the processes taking turns to enter their CRs, using one variable for shared lock, and using a set of variables for shared warnings.

Key Steps for Peterson’s Solution

Enter\_region(process #)

Critical\_region()

Leave\_region

Noncritical\_region()

What is the problem with Peterson’s solution to the race condition problem?

Peterson’s solution is correct, but it still requires busy waiting. When a process wants to enter its CR, it checks if it is allowed. If not, the process has to wait.

Solutions- the priority inversion problem.

What is the problem with the Simple Solution to the Producer-Consumer Problem?

The simple solution may lead to a race condition. The essence of the problem is that a wakeup signal sent to a process that is not yet sleeping is lost.

Quick fix- add a wakeup-waiting bit to each process.

Two operations of a semaphore- usually calledown() and up() (generalizations of sleep() and wakeup(), respectively)

Down operation checks to see if value > 0, if so decrements the value by 1 and continues to run, if not the process is put to sleep.

Up operation increments the value of the semaphore addressed. If an up operation on a semaphore that was positive, simply increments the value by 1. If 0, only one process sleeping on it, the semaphore value is increased to 1 and the process is awaken. If 0, and 2 or more processes were sleeping, then the semaphore value will still be 0 but the # of processes sleeping on it decreases by 1.

**Ch 2. Scheduling**

When to Schedule Processes

Scheduling decisions are made when

1. 1.A new process is created – parent or child to run first?
2. 2.A process exits – select one from the set of ready processes
3. 3.A process blocks on I/O, on a semaphore, or for some other reason, another process has to be selected to run

How to measure the performance of a scheduling algorithm? Calculate the average turnaround time. SJF is optimal in terms of shortest average turnaround.

Robin-Robin Scheduling- Each process assigned a quantum of time. Simple, fair and widely used. Design issues: 1. Size of quantum, 2. Cpu overhead, and 3. Response time.

Which process should be assigned higher priority?

Low-priority processes may never get a chance to run if processes with high priority are always in the ready state, how is this issue resolved? Decreasing the priority of high priority processes at each clock tick (or cpu cycle). Each process assigned a quantum of time and scheduling next higher-priority process to run after quantum expires.

Process Scheduling examples: Four processes (A,B,C, and D) arrive at a computer center at almost the same time. They have estimated running times of 10, 5, 2, and 7 minutes with priorities of 2, 4, 1, and 3. 4 being the highest.

Round Robin A[10] B[5] C[2] D[7]. Finish time for A is 24, finish time for B is 17, finish time for C is 6 and finish time for D is 22. Avg is 69/4.

Priority Scheduling (order of priority, 4, 3, 2, 1. B, D, A, C) B[5] D[12] A[22] C[24] = 5 + 12 + 22 + 24 = 63/4.

First-come, first-served (run in order of A, B, C, and D) 10 + (answer from 5+10) 15 +(answer from 15+ 2) 17 + (answer from 17 + 7) 24 = 66/4.

Shortest job first- avg = 2 + 7 + 14 + 24 = 47/4.