CSE 2221 SYSTEM(S) PROGRAMMING

Introduction to system programming.

- System(s) programming / system programming is the activity of programming computer system software.
- The primary distinguishing characteristic of system programming when compared to app programming is that application programming aims to produce sw which provides services to the user directly e.g wordprocessors, information management systems etc. whereas systems programming aims to produce software and software platforms which provide services to other software eg. operating systems, computational science applications, game engines, videogames, industrial automation etc.
- System programming requires a great degree of hardware awareness its goal is to achieve efficient use of available resources either because the software itself is performance critical eg videogames or because even small efficiency improvement directly transform into significant monetary savings for the service provider.
- The following attributes characterize system.
  1:- Programmer can make assumptions about H/w and other properties of the system that the program runs on and will often exploit these properties for example by using an algorithm that is known to be efficient when used with specific H/w.
  2. Usually a low level programming language is used so that
     a) Programs can operate in resource constrained environment.
     b) Programs written to be efficient with little runtime overhead
     c) Programs may use direct control over memory access and control flow.
     d) Programmer may write parts of the program directly in assembly language.
  3. Often systems programmer cannot be run in a debugger.

In system programming often limited programming facilities are available the use of automatic garbage collection is not common and debugging is sometime hard to do.

N/B: Garbage collection - process of finding and deleting objects which are no longer being referenced by other objects in other words garbage collection is the process of removing any objects which are not being used by any other objects.

Implementing certain paths in operating systems and networking requires system programming example:- Implementing paging (virtual memory) or a device driver for an OS

A Programming Illusion.

- System programs can sometimes be written to ~~external~~ extend the functionality of the operating system itself and provide functions that higher level application can use.
- The following program use the API of the UNIX os
- The following programs gets its input from the keyboard or a disk file and writes its output to display screen or to file on the disk. Such programs are called console applications Because the keyword and the control screen are part of the control devices.
- UNIX automatically creates the standard input and standard output devices for it, opens them and makes them ready for reading and writing respectively.
- In C (and C++), stdin and stdout are variables in the (preprocessor <stdio.h> header file that relates to the standard input and standard output device respectively.
- By default, the keyboard and the display of the associate terminal are the standard input and output devices respectively.

1. Examples.

C program using a simple I/O model.

```
#include <stdio.h>
/* copy from stdin to stdout */
int main ( )
  {
  int c;
  while ((c=getchar( ))!= EOF)
    putchar (c);
      return 0;
    }
```

→ end of file.

2. Simple C++ program using I/o model.

```
#include <iostream>
  using namespace std;
/* copy from stdin to stdout using C++ */
  int main( )
  {
   char c;
   while ((c = cin.get( )) && ! cin.eof( ))
   cout . put (c);
   return 0;
  }
```

cin >> ⟶ cin.get.

cout >> ⟶ cout.put.

Those programs give us the illusion that they are directly connected to the keyboard & the display device via C library functions; getchar( ) and putchar( ); and the C++ iostream member functions get( ) and put( ). Either of them can be run on on a single user desktop computer or on a multiuser time shared workstation in a terminal window and the results will be same

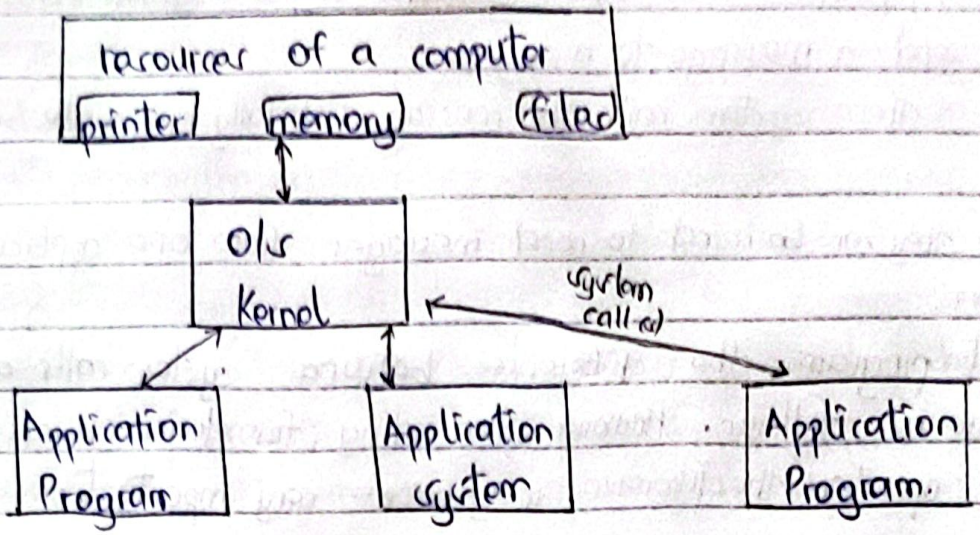
Process:

- A program is an executable file, and a process is an instance of a running program.
- When a program is run on a computer, it is given variable resources such as a primary memory space, mappings of various kinds and privileges such as the right to read or write certain files or devices.
- As a result at any instance of time, associated to a process is the collection of all resources allocated to the running program as well as any other properties and settings that characterise that process such as the value of the processors registers.
- UNIX assigns to each process a unique number called its process-ID or PID for example; at a given instant of time, several people might all be running the GNU C compiler, gcc.
- Each separate execution instance of gcc is a process with its own unique pid.
- The ps command can be used to display which process ~~with its own unique pid~~
- ~~The ps~~ are running and various options to it control what it outputs
- At the programming level, the function getpid() returns the process id of the process that invokes it.
- The following program shows how it is used; we will also see that getpid() is an example of a system call

System Call

```
#include <stdio.h>
#include <unistd.h>
int main()
{
printf("I am the process with process-id %d\n", getpid())
return 0;
}
```

System calls.



An O/s is a program with two jobs: it manages the resources of the computer (files, memory, peripherals etc) and it manages all other programs running on the computer.

The programs we write access the resources of the computer by calling functions executed by the kernel.

This functions are called system calls.

A collection of system calls is sometimes referred to as API.

There are dozens of system calls supported by any linux or unix kernel. Some common examples include (non of these list is all inclusive)

① File I/O : open( ), read( ), write( ), close( ), creat( ), link( ), lseek( ).

This system calls ask the O/s to access a file

② access control : fork( ), execl( ), execv( ), wait( ), system( )

This system calls ask the O/s to run another program or control how it runs.

③ Memory management : mmap( ), brk( ) - This system calls ask the O/s to provide memory to be used by the application program.

④ Time management : time( ), setitimer( ), settimeofday( ), alarm( ). This system calls ask the O/s to access the system clock in some cases taking an action that affects a program when a given time interval has passed.

Interprocess communication (IPC):
signal( ), pause( ), kill( ), sigaction( ). This system calls asks the OS to send a message to a program.
For most of these system calls the messages usually originate in the kernel.
Although they can be used to send messages from one application program to another.
For most programs the differences between system calls and library calls are as follows: However when coding for device drivers or other low level operation the differences may become very important.

Standard library functions are built on top of system calls.
example malloc( ) and related functions are built on top of the mmap( ) and brk( ) system calls, meaning they call them to get the job done.
The maloc function is in the C standard library while mmap is a system call.
This means that greater control over the computer resources can be obtained through system calls.
standard library functions tend to be more standardized than system calls They have both standards bodies that is; ANSI defines library standards functions while the POSIX defines system call standards however since the system calls provide direct access to the kernel different OSs (including different variations of UNIX/LINUX will have at least slightly different system calls.
System calls cause a "context switch"
Meaning that the application program stops while the kernel program runs for a while to perform the requested operating.
Because of this a system function call takes longer than a normal function call within your own program.
Standard library calls often optimize operations to minimize the number of system calls thus speeding up program execution.

1) What is a system call; why would a program make one.
2) What are the differences between a system call and a library function call.
3) Is malloc ( ) a system call or a library call.
4) What are the differences between open ( ), read ( ), write ( ), close ( ) and fopen ( ), fread ( ) and fclose ( ).

NIB in computing a system call is how a program requests a service from an o/s kernel; thus may include hardware related services eg accessing the hard disk ( ), creating an executing new processes and communicating with integral kernel services like scheduling.

System calls provide an essential interface between a process and the Operating system.

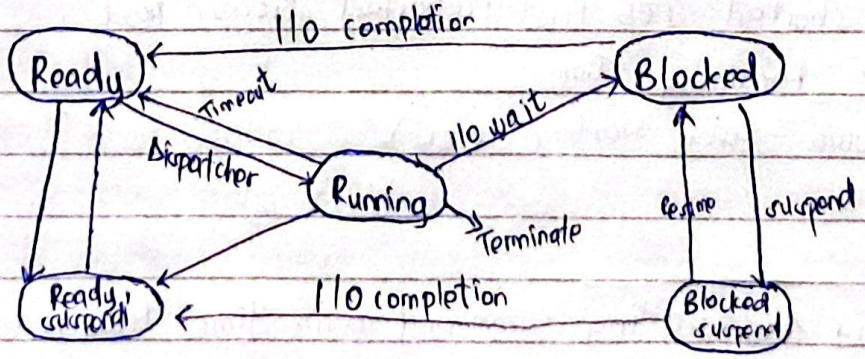
Pro

PROCESS MANAGEMENT AND SCHEDULING.

The major responsibilities of a multitasking O/s is process management The OS must allocate resources to processes enable processes to share and exchange information protect the resources of each process from other processes and enable synchronization amongst processes.

A process can be defined as a program in execution it can be defined as a program currently making use of the processor at any one time.

The diagram below shows the various state of a process.



PCB - process control Block. -u
- Has process state
- Program counter

s COMMUNICATION (IPC)

nterfaces that allow programmers to communicate
and allow programs to run concurrently.
chieved using the following intercommunication

ata to flow from one process to another only in
ally from the output of one process to the input of

communication is made possible using 2 pipes, 1 in
ch must have a common process origin.
utput process to the input process is buffered.
rocess receives it
g data from a program and passing it onto

document from the computer system where re
tem.

s a specific name.
processes that do not have a shared
.
un as First in First Out (FIFO).

processes to pass messages between
ingle or several message queues
ges each message queue by giving
ge on the queue.
created by one process and then used
ead or write messages to the queue.
r processes) create message queues

- CPU register such as index register.
- Condition codes.
- accumulators.
- Accounting information such as amount of CPU and real time used.
- Memory management info eg Base and index registers
- Input/Output status information

Process scheduling.

1. long term scheduler
2. medium-term scheduler
3. short-term scheduler.

Scheduling Policies

1. Non pre-emptive
2. Pre-emptive.

Pre-emptive scheduling permits a process to be removed from the processor when other high priority processes come in. but non preemptive scheduling cannot permit this it works on a first come first serve its fairer though it keeps short jobs waiting.

Scheduling algorithm.

1. FIFO.
2. RR - Round Robin.
3. SJF / SPN - shortest Job first / shortest Process Next
4. SRT - shortest Remaining Time.
5. Higher Response Ratio Next (HRRN)

HRRN what is the waiting time and service time briefly explain and give an example.

What are the scheduling objectives.

INTER - PROCESS COMMUNICATION (IPC)

It is a set of interfaces that allow programmers to communicate between processes and allow programs to run concurrently.
This communication is achieved using the following intercommunication techniques

1. Pipes - allow data to flow from one process to another only in one direction typically from the output of one process to the input of another.
2. - Although 2 way communication is made possible using 2 pipes, 1 in each direction which must have a common process origin.
The data from the output process to the input process is buffered until the input process receives it.

Examples :- i) Copying data from a program and pasting it onto another.

ii) Printing a document from the computer system where re data output from the system.

2. Named pipe(s).
In this case the pipe has a specific name.
It can be used by processes that do not have a shared common process origin.
Named pipes are also known as First in First Out (FIFO).

3. Message Queuing.
Message queues allows processes to pass messages between themselves using either a single or several message queues.
The systems kernel manages each message queue by giving an identity to each message on the queue.
A message queue may be created by one process and then used by multiple processes that read or write messages to the queue.
Application programs (or their processes) create message queues

leaved execution provides major benefits in processin
program structuring.
sor system it is possible not only to interleave
lo overlap them.
and overlapped processes can be viewed as examples
ocesses, they both present the same process.
d of execution cannot be predicted it depends on
other processes, the way the operating system handles
scheduling policies of operating system.

obal resources. if 2 processes both make use of
e and both perform reads and writes on that
e order in which the various reads and writes are
al.
r the OS to manage the allocation of resources

ient for the OS to simply lock the channel and
other processes.
difficult to locate a programming error, because
ally reproducable.

OS have to regulate the order in which
ta is known as synchronisation
sues
method of ensuring that when one process is
esource (file or variable), the others are excluded
e.
ssing co-operating so that only one of the
ss to a shared resource at a time.

and send and receive messages using API.

4. Semaphores.
They are used to synchronize events between processes. Th
They are integral values greater than or equal to zero.

5. Shared memory.
This allows processes to interchange data through a defined area
memory. example1 process would write to an area of memory and
another could read from it.
To do this writing process must check to see that the read process is
reading from the memory at the time and vice versa.
If there are occuring the other process must wait for the other process
to complete.
This is ~~documented~~ implemented using semaphores where only one process is allowed to
access memory at a time.

6. Sockets.
These are typically used to connect over a network between a client
and a server.
Although peer to peer connections are also possible.
Sockets are end point of a connection and allow for a standard
connection which is computer and OS dependent.

CONCURRENCY.
- A state in which processes) exists simultaneously with another
process and this is said to be concurrency.
- Principles of concurrency
In a single ~~multi~~processor multiprogramming system processes are
interleaved in time to yield appearance of simultaneous execution
even though actual parallel processing is not achieved and
certain overhead is involved in switching back and forth between

processes, interleaved execution provides major benefits in processing efficiency and program structuring.
In a multiprocessor system it is possible not only to interleave processes but to overlap them.
Both interleaved and overlapped processes can be viewed as examples of concurrent processes, they both present the same process.
The relative speed of execution cannot be predicted it depends on the activity of other processes, the way the operating system handles interupts and scheduling policies of operating system.
Following dif
→ sharing of global resources: if 2 processes both make use of a global variable and both perform reads and writes on that variable then the order in which the various reads and writes are executed is critical.
→ It is difficult for the OS to manage the allocation of resources optimally.
- It may be inefficient for the OS to simply lock the channel and prevents its use by other processes.
- It becomes very difficult to locate a programming error, because reports are not usually reproducable.

Synchronization.
The process in which OS have to regulate the order in which processes access data is known as synchronisation.

Synchronization issues.
1) Mutual exclusion - method of ensuring that when one process is accessing a shared resource (file or variable), the others are excluded from doing the same.
- Two or more processing co-operating so that only one of the processes has access to a shared resource at a time.

2. Race Condition.

- When two concurrent processes are sharing the same storage area.
- A condition in which the behaviour of two or more processes depends on the relative rate at which each process executes its program.

3. Critical Section.

The mechanism for preventing two processes from accessing the same critical area.

A segment of code that cannot be executed while some other process is in a corresponding segment of code.

The fundamental is two or more processes can cooperate by means of simple signal, such that a process can be forced to stop at a specified place until it has received a specified signal.

Any complex coordination requirement can be satisfied by the appropriate structure of signals to transmit a signal via semaphores a process executes a primitive signal(s).

To receive a signal via semaphores a process executes if the corresponding signal has not yet been transmitted the process is suspended until transmission takes place.

To achieve the desired effect we can view the semaphore as a variable that has an integer

① A semaphore may be initialised to a non negative value.

Weight operation decrements a semaphore value.

If value becomes negative then the process executing the weight is blocked

The signal operation increments the semaphore value.

Semaphores are operated on by a signal operation which increments the semaphore value and the weight operation.
The initial value of a semaphore indicates the number of weight operations that can be performed on the semaphore thus

$$V = I - W + S$$

where I = Initial value of semaphore.
W = number of completed weight operations performed on the semaphore
S = Number of signal operation
V = current value of semaphore which must be greater than or equal to zero
as V is ≥ 0 then $I - W + S \geq 0$ which gives

$$I + S \geq W$$

or

$$W \leq I + S$$

Thus number of weight operations must be less than or equal to the initial value of semaphore plus number of signal operation.
Binary semaphore will have an initial value of 1 ($I = 1$) thus:

$$W \leq S + 1.$$