Operating System

* Definition : A program that acts as an intermediary between users of a computer system and hardware
* Goals :
* Resource allocator : Manages all resources to ensure they are shared in an efficient and fair manner
* Decides which app gets how much resources and when
* Control executions of other programs/app to prevent er­­rors and improve usability
* Enabler of communication/coordination between apps
* Gives and illusion of infinite resources to app
* Control flow : Applications, Operating System, Kernel, Hardware
* Kernel : (Part of OS) collection of functions that provide essential functionalities of OS i.e, managing memory, network, file, processes, system calls
* OS handles these components (shell,GUI,system tools,services)
* Kernel is interrupt driven (Interrupt is a signal instruction generated by hardware/software)
* Interrupt :
* Hardware : external(i/o devices), internal(memory access violation) : generated by external devices or internal hardware unit
* Software : trap(syscall), exception(divide by 0) : genrated by running program
* I/O devices and CPU can execute in parallel
* Each device controller is in charge of a particular device type
* Each device controller has a local buffer
* Data transfers between local buffer and main memory
* Device controller sends an interrupt to the CPU to indicate I/O completion
* Spool (Simultaneous peripheral operations online) : Is a buffer that holds output for an I/O device. A part of disk can actually be used as a spool
* Spooling is a special form of multiprogramming
* Multiprogramming : multiple jobs loaded into memory at the same time and job scheduler selected a job to assign a cpu
* For multiprogramming to work : a good mix of CPU bound and I/O bound jobs…….so that job a assign to i/o and then b assign cpu then …..
* Multitasking (timesharing) : logical extension of multiprogramming that is CPU switches jobs so fast that users can interact with each job while its running
* Process can execute in 2 modes
* User mode : run normal tasks
* Kernel mode (privileged mode) directly talk to CPU/peripherals to schedule tasks
* System call or interrupt changes mode to kernel
* System call : mechanism used by an application program to request service from OS
* Interrupt :
* Synchronous : basically execptions i.e, divide by zero, bad pointer derefernce, app wants some services(syscall)
* Asynchronos : by external agent i.e, i/o devices, timer
* Interrupt handler : What happens if a program issues "int "?
* Control jumps to the kernel at a prescribed address (the interrupt handler)
* Address of interrupt handler found using the IDT (index num)
* The register state of the program is saved by the kernel
* Interrupt handler runs and handles the interrupt
* When handler completes, resume program
* System calls
* Provide an interface to the services made available by an OS
* Use API
* Fork, pipe, execvp etc are example
* Function we write(printf,scanf) -> lower level standard C library call (read, write) -> syscall instruction
* Process control i.e, fork(),exit(),wait()
* File management i.e, open(),close(),read(),write()
* Device management i.e, ioctl(),read(),write()
* Interprocess communications i.e, pipe(),kill(),signal()
* Protection i.e, chmod(),umask()
* Process : Process is a program in execution
* Program is a passive while process is a active entity
* Program becomes process when code is loaded in memory and ready to execute
* Job scheduling is long term scheduler i.e, loaded jobs into memory
* Process scheduling is short term scheduler i.e, allocated CPU to jobs
* States of a process :
* New : process is being created
* Ready : process is waiting to be assigned to a processor
* Running : instructions are being executed on CPU
* Waiting : process is waiting for some event to occur
* Terminated : process has finished execution
* See pg no 27 of slide 4 for detailed transition states diagram
* Process control block (PCB) : each process is represented in kernel as a PCB
* Structure of a PCB :
* Process state : running, waiting etc
* Program counter : location of instruction to next execute
* Content of CPU registers
* CPU scheduling information : priorities, scheduling queue pointers
* Memory management info : base and limit registers, page tables
* Accounting info : CPU used, clock time elapsedsince start, pid
* I/O info : I/O devices, allocated to process, list of open files
* Context Switch : kernel saves the state of the old process and loads the saved state for the new process via a context switch. Context of a process == PCB
* Dispatcher: The kernel process that assigns the CPU to a process selected by the short-term scheduler
* A = Fork(), A>0 return pid of child process, A=0 return zero to the newly created child process, A<0 child creation unsuccessful
* Fork() : creates new process, exec() : replace new process’s memory with new code
* Two variants : • wait() - suspends execution of the current process until one of its children terminates • waitpid(pid) - suspends execution of the current process until the child specified by pid terminates
* Check pg no 25 of slide 5 for process creation and exec example code
* If no parent is waiting, then zombie process • If parent terminated without invoking wait() then orphan process
* An **orphan process** is a computer **process** whose parent **process** has finished or terminated, though it (child **process**) remains running itself. A **zombie process** or defunct **process** is a **process** that has completed execution but still has an entry in the **process** table as its parent **process** didn't invoke an wait() system call.
* Inter-process communication (IPC) : processes executing concurrently in OS may be independent or cooperting. Ways to do IPC : shared memory ( shmget(), shmat(), shmdt(), shmctl())
* Check pg no 40 of slide 5 for code of function calls
* Need of scheduling :
* CPU burst : process is being executed in the CPU
* I/O burst : process is waiting for I/O to be done
* Process alternates between CPU and I/O burst
* Maximum CPU utilization obtained with multiprogramming­
* Long term scheduler controls degree of multiprogramming
* Pre-emptive scheduling : CPU can be forcibly taken away from running process ie, timer interrupt…..results in race condition
* Non-preemptive scheduling : only when processes switched from running to waiting or when process terminates
* Throughput : no of processes that complete their execution per time unit (on average)
* Turn around time : amount of time to completely execute a process
* Waiting time : amount of time process spends waiting in the ready queue
* Burst time : amunt of time a process is executed on CPU
* Response time : amount of time from when a process is submitted to the time when the first response is produced
* Scheduling algorithms :
* First come first serve (FCFS) :
* Non preemptive scheduling
* Ready list is maintained as a FIFO queue
* Issue : average waiting time is long
* Convoy effect : process with large CPU burst can delay several processes with shorter CPU bursts.
* Shortest job first (SJF) :
* Non preemptive
* SJF algo minimizes the average waiting time
* Estimate next CPU burst time : Tn+1 = aTn + (1-a)Tn ; Tn nth CPU burst
* Shortest remaining time first scheduling is pre-emptive version of SJF
* Priority scheduling :
* Can be preemptive or non preemptive
* Issue : low priority processes may never execute
* Resonse ratio(RN)=1+(time since arrrival)/(CPU burst time)
* Priority detemine by nice value : -20 high, 19 low, 0 default
* Round robin scheduling :
* For time sharing system i.e, time slice defined here
* Ready queue is a circular queue in this case
* CPU goes around each process in ready queue and execute for 1 time slice (delta)
* Timer is set to interrupt CPU at the end of each time slice
* Multi level queue scheduling :
* Readyqueue is partitioned into separate queues ie. Foregrund (interactive) and background (batch)
* In between queues serve all from foreground then background
* Each queue gets certain amount of CPU time ie, 80% to foreground in RR, 20% to background in FCFS
* Multilevel feedback queue scheduling :
* We allow processes to move between queue
* Long running processes may starve which move to FCFS. So to overcome this we do periodic priority boost and priority boost with aging.
* Multithreading :
* Thread : Process is a program in execution with a single thread of control
* Dgaddgd
* Gdgdfgdf
* Gdfgdfgfdgdf

OS LAB

**int shmget(key\_t** *key***, size\_t** *size***, int** *shmflg***);**

shmget - allocates a System V shared memory segment

**IPC\_CREAT**

Create a new segment.

0700 : Read write execute permission file owner ke liete

void \*shmat(int *shmid*, const void \**shmaddr*, int *shmflg*);

The *shmat*() function attaches the shared memory segment

associated with the shared memory identifier specified by *shmid*

to the address space of the calling process.