OS-5

1] Describe process creation with algorithm.

→ ① Process creation :-

① The only way to create a new process is unix is to use the fork system calls. The process which calls fork is called the parent process & the newly created process is called child process.

② pid = fork();

③ pid for parent process is the process ID of child process. And pid for child process is 0. The process 0 is the only process which is not created via fork.

④ Steps followed by kernel for fork :-

i) It creates a new entry in process table

ii) It assigns a unique ID to the newly created process.

iii) It makes logical copy of regions of parent process. If a regions can be shared, only its reference & count is incremented instead of making a physical copy of region.

iv) The reference counts of file table entries & inodes of process are increased.

v) It turned the child process ID to parent & 0 to the child.

Algorithm :-

Algorithm: fork
Input: none
output: to parent process, child PID number
to child process, 0

{
check for available kernel resources;
get free proc table slot; unique PID number;
check that user not running too many processes;
mark child state "being created";
copy data from parent proc table slot to new child slot;
increment counts on current directory inode and changed root (if applicable);
increment open file counts in file table;
make copy of parent context (u area, text, data, stack) in memory;
push dummy system level context layer onto child system level context;
//dummy context contains data
// allowing child process to
//recognize itself & start running
// from here when scheduled
if (executing process is parent process)
{
if (executing process is parent process)
{
change child state to "ready to run";
return (child ID); //from system to user
}
else //executing process is child process

```
{
    Initialize u-area timing fields;
    return (0);
}
```

Q.2] Explain algorithm for recognizing signals & handling signals.

① signals inform processes of the occurence of asynchronous events.

② Handling signals:-

i) The kernel handles signals in context of process that receives them so a process must run to handle signals.

ii) old function = signal (signum, function);

ⓐ where signum is the signal number → the process is specifying the action for

ⓑ function is the address of user function the process wants to invoke on receipt of signal and

ⓒ return value oldfunction was the value of function in most recently specified call to signal for signum.

ⓓ The process can pass the values 1 or 0 instead of function address :

— The process will ignore future occurence of signal if the parameter value is 1 &

— exit in kernel on receipt of signal if its value is 0.

ⓔ The u-area contains an array of signal-handlers fields, one for each signals defined in system. The kernel stores the address

of the user function in field that corresponds to signal number.

③ Alg :-

Algorithm: psig
Input : none
output: none

{
get signal number set in process table ent
clear signal number in process table entry;
if (user had called signal call to ignore this signal)
return ; //done
if (user specified function to handle the signal)
{
get user virtual address of signal catche stored in U area;
//the next statment has undesirable side-effects
clear u area entry that stored address of signal catchee ;
modify user level context:
artifically create user stack frame to mimic call to signal catcher function;
modify : system level context:
write address of signal catcher into program counter field of user saved register context;
return ;
}
if (signal is type that system should dump core image of process)

}
create file name "core" in current directory;
write contents of user level context to
file "core";
}
invoke exit algorithm immediately;
}

④ If signal handling function is set to default value, the kernel will dump a "core" image of process for certain types of signals before exiting.

⑤ If process decides to ignore, the signal field is not reset & the process will continue ignoring the signal.

Q.3] Explain exit system call with algorithm.

→ ① Processes on UNIX system exit by executing the exit system call.
② When a process exits, it enters the zombie state, releases all of its ~~res~~ resources and its context except for its process table entry.
③ exit (status);
— where status is the exit code returned to parent.
— The process may call exit explicitly, but the startup routine in C calls exit after the main function returns.
④ Algorithm: exit
Input: return code for parent process
output: none.

```
{
  ignore all signals;
  if (process group leader with associated
  control terminal)
  {
    send hangup signal to all members of
  the process group;
    reset process group for all members to
    0;
  }
  close all open files (internal version of
  algorithm close);
  release current directory (algorithm: input);
  release current (changed) root, if exists
    (algorithm: input);
```

free regions, memory associated with process (algorithm : freereg);
write accounting record;
make process state zombie;
assign parent process ID for all child processes to be Init process (1);
if any children were zombie, send death of child signal to init;
send death of child signal to parent process;
context switch;
}

Q.4] Describe awaiting process termination.

→ (1) A process can synchronize its execution with the termination of child process by executing the wait system call.
(2) pid = wait (stat_addr);
— where pid is process ID of zombie child, and stat_addr is the address in user space of an integer that will contain the exit status code of child.
(3) The kernel does not "long jump" from sleep, but returns wait.
(4) The kernel will restart the wait loop, find zombi child at least one is guaranteed to exist, release the child's process table slot and return from wait system call.

(5) If the process catches "death of child" signals, the kernel arranges to call the user signal-handler routine, as it does for other signals.

(6) If the process ignores "death of child" signals, the kernel restart the wait loop, frees the process table slot of zombie children and searches for more children.

Q.5] Explain algorithm for booting the system.

→ ~~Algo:~~

```
/*
Algorithm : start
Input : none
output: none
*/
{
  initialize all kernel data structures;
  pseudo-mount of root;
  hand-craft environment of process 0;
  fork process 1:
  {
    //process 1 here:
    allocate region;
    attach region to init address space;
    grow region to accommodate code about
     to copy in;
    copy code from kernel space to init
    user space to exec init;
    change mode: return from kernel to
                 user mode;
   /* init never gets here -- as result of
     above change mode,
    * init exec's /etc/ init & becomes a
    "normal" user process
   * with respect to invocation of system
     calls
   */
  }
  // proc 0 continues here
   for kernel processes;
   /* process 0 invokes the swapper to manage
* the allocation of process address space
* to main memory & swap devices.
```

* This is an infinite loop; process o
* usually sleeps in the loop unless there is work for it to do.
*/
execute code for swapper algorithm;
}

Q.6] write short note

a] User id of process

→ ① There are two user ID associated with a process, the real user ID & the effective user ID or setuid (set user ID).
② The real user ID identifies the user who is responsible for running process.
③ The effective user ID is used to assign ownership of newly created files to check file access permissions and to check permission to send signals to processes via kill system call.
④ setuid (uid);
— where uid is new user ID, and its result depends on current value of effective user ID.
— If the effective user ID of calling process is superuser, the kernel resets the real & effective user ID fields in process table & u-area to uid.
— If its not superuser, the kernel resets the effective user ID in the u-area

to uid, if uid has the value of real user ID or if it has value of the saved user ID. otherwise, the system call return an error. Generally, a process inherits its real & effective user ID from its parent during the fork system call & maintains their values across exec system calls.

b] shell :

① A shell provides you with an interface to Unix system.
② It gathers input from you & executes programs based on that input.
③ When a program finishes executing, it displays that program's output.
④ shell is an environment in which we can run our commands, programs & shell scripts
⑤ If the shell recognizes the input string as a built-in command, it executes the command internally without creating new process; otherwise, it assumes the command is the name of an executable file.
for ex. → ls | wc

Tutorial No.7

Q.1] explain algorithm for process scheduling.

→ ① Algorithm: schedule-process
input : none
output : none
{
while (no process picked to execute)
{
for (every process on run queue)
pick highest priority that is loaded in memory ;
if (no process eligible to execute)
idle the machine ;
/* interrupt takes machine out of idle state */
}
remove chosen process from run queue ;
switch context to that of choosen process,
resume its execution ;
}

② ① The kernel executes the algorithm to schedule a process.
② selects the highest priority process from those in the states "ready to run & loaded in memory" and "preempted."
③ It does not select a process that is not in main memory.
④ If several processes tie for highest priority, the kernel picks the one that has been "ready to run" for longest time.

Q.2]

⑤ If there are no processes eligible for execution, the processor idles until the next Interrupt.

Q. 2] Describe scheduling parameters used in process scheduling.

→ ① Each process table entry contains a priority field for process scheduling.
② The priority of process in user mode is a function of its recent CPU usage.
lower priority if a process has recently used the CPU.
③ The range of process priorities are partitioned into two classes:
User priorities and kernel priorities.
④ each class contains several priority values. each priority has queue of processes logically associated with it.
⑤ User level priorities are below a threshold value and kernel-level priorities are above the threshold value.
⑥ Processes with user level priorities were preempted on their return from the kerel to user mode.
⑦ Processes with kernel-level priorities achived them in the sleep algorithm.
⑧ kernel-level priorities are further subdivided:
i) Processes with low kernel priority wake up on receipt of signal, but processes with

high kernel priority continue to sleep.

ii) The priorities called "swapper", "waiting for disk I/O", "waiting for buffer", and "waiting for inode" are high, noninterruptible system priorities.

3] Explain various system call for time.

→

4] Describe fun^n. of clock interrupt handler.

→ ① The function of clock interrupt handler are to
— restart the clock
— schedule invocation of internal kernel functions based on internel timers.
— provide execution profiling capability for the kernel and for user processes
— gather system & process accounting statistics
— keep track of time
— send alarm signals to processes on request
— periodically wake up the swapper process
— control process scheduling

② some operations are done every clock interrupt, whereas others are done after several clock ticks.

③ At every clock interrupt, the clock handler decrements the time field of first entry.

④ If the time field of first entry is less than or equal to 0 then, the specified function should be invoked.
— the clock handler does not invoke the fun^n directly.

⑤ The clock handler schedules the fun^n by causing a "software interrupt".
— because software interrupts are at lower priority level than other interrupts they are blocked until kernel finishes

handling all other interrupts.
— clock interrupts could occur until the software interrupt occurs and, therefor, the time field of first entry can have a negative value.