QUIZ - 1

1) System call

2) Real time Operating System

3) code
code
code

4) ~~kkk~~ ready

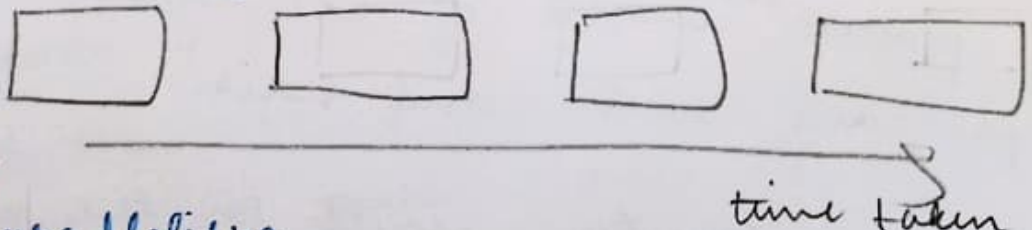
5) Memory

6) child finishes the execution. or some process finishes the execution.

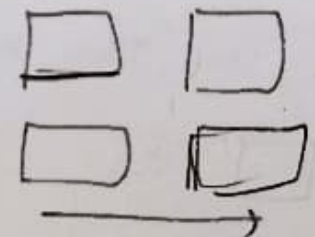
7) ~~one to one~~ many to many

8) Medium - term - scheduler

9) concurrency

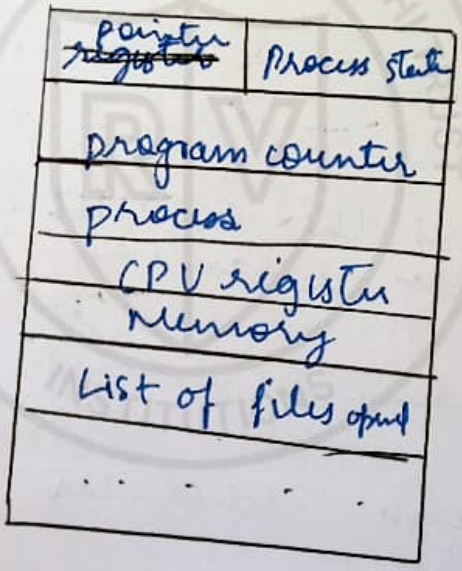


Parallelism



1) Process is a part of program in execution. Process control Block or transition control block is way of represent each process state as the state of the processes key switch.

Process Control Block



Each process state consists of CPU register its program counter and stack.

Program Counter: waits for the process to be allocated in the memory

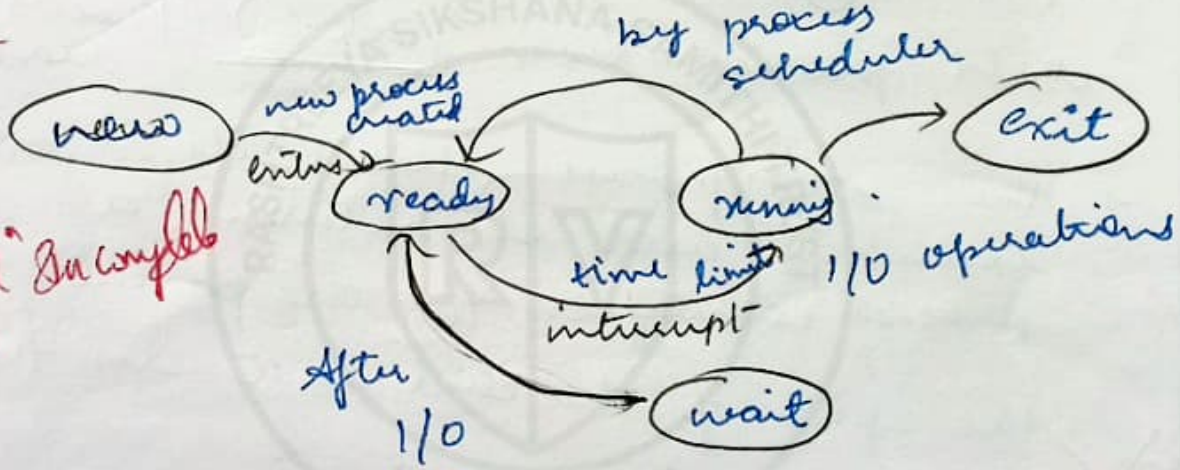
→ new : When the process is ~~starting~~ ~~to be executed~~ created

→ ready : It is ready for execution.

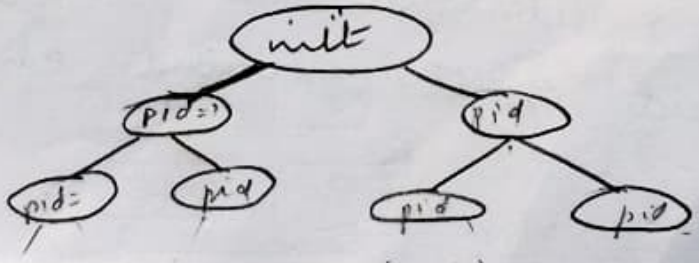
→ Running : A process which is running

→ wait : It is to wait for the child to be executed and then parent starts the process

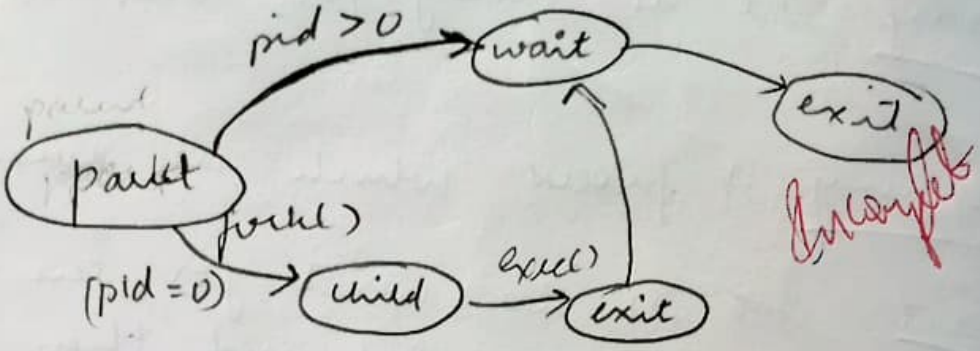
6



Here the parent process is called as the init in which each process has its own identifier (pid=0) for ~~parent~~ and then there are subprocesses which have their own process identifier



2) Lifecycle of a child process



→ Here when the parent creates a new child process uses fork() (Process creation)

→ the child process starts its execution and it returns a process identifier to the parent and the pid of the child is 0 in this case

→ After When the child is executing the parent is in wait state and waits till the child finishes and uses wait()

wait(): Returns the memory address & the pid of the executed child.

The parent then terminates and the process is completed.

exec(): This command makes the child program to start its execution

5b) Given

$S = 30\% = 0.3$

$P = 1 - 0.3 = 0.7$

$N = 5$

$$\frac{1}{S + \frac{P}{N}} = \frac{1}{0.3 + \frac{0.7}{5}} = \frac{1}{0.44}.$$

3

speedup ≤ 2.2727 //

The maximum speedup is 2.2727.

2) ZOMBIES and ORPHANS

A zombie state is the state in which the child is executed but the memory allocated in the Process control Block is not realased since the parent did not execute wait()

→ It isn't in zombie state for long It can again call wait(&status) with the memory address of the

executed child and deallocate it so that the parent gets terminated

ORPHANS

Child process is present and the parent is terminated.

3) True,

i) Time sharing model is a logical extension of the CPU in which along with multiprogramming it helps in multitasking.

There is Job scheduling and CPU scheduling

In Job scheduling the Job scheduler chooses the required processes to the memory, if there are more process than the memory spall, the Job Scheduler needs to choose

In CPU scheduler when all the process are ready for execution which process should start is decided by this.

Multiprogramming is a technique to maximize CPU utilization and it happens so because the CPU keeps switching from one process to another which given an illusion of concurrent.

so when time slice exceeds CPU burst

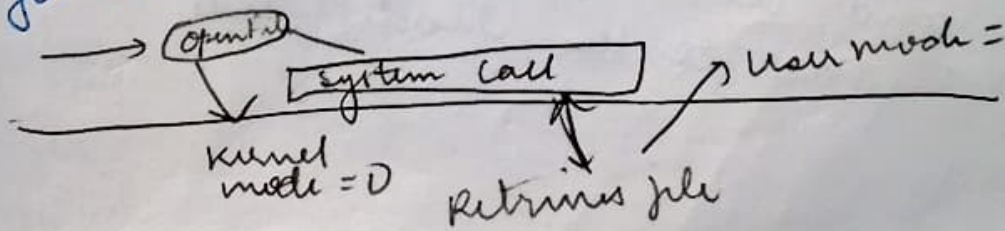
Time sharing systems are identical to multiprogramming systems

True,

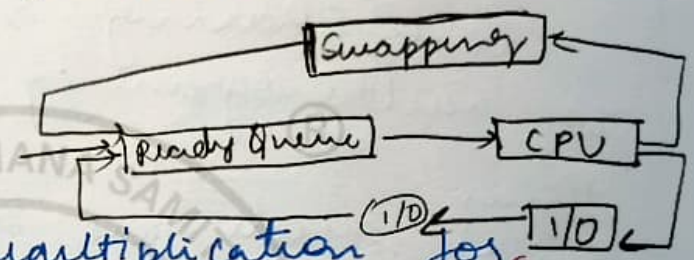
(ii) Kernel mode is an essential part of dual mode which provides only priviliged access,

When the user opens a file, the system which is in the user mode switches to the kernel mode which implements system call and acquires the file from the driver and returns back to the user mode.

Kernel mode is only accessed by the operating systems and has restricted access, Hence it is essential for it to be isolated & protected executin



(iii) Swapping Term: process in & out of the memory space increases OS overhead Since there are fixed number of kernel threads and everytime it needs to be swapped new kernel thread needs to be created which increases the memory space & hence the OS overhead decreases



Two modes:
user mode
kernel mode.

(iv) To do matrix multiplication for a uniprocessor environment it is required to use an efficient algorithm since,

in uniprocessor environment it only executes one process at a time

So, it is essential to have an efficient algorithm for faster program.

(i) Also the ready state when swapped out pushes a new process everytime to the CPU and it restacks the unexecuted process back to the ready state

(iv) Since it is a uniprocessor it executes only one process so we can use more threads to do matrix multiplication.

~~A~~ threads access the same ~~thing~~ memory space as other threads it is a basic unit of execution.

4 a)

(i) User convinience:

Partition based resource allocation they are distributed among some threads

Pool based resource allocation is where pool of resources are waiting to be allocated in a resource.

The partition based resource allocation increases the convinience of the user than pool based resource allocation given the time is reduced and is distributed.

(ii) Efficiency of use: Pool once partition since it utilizes the CPU scheduling efficiency.

4b) User level threads

→ User level threads are executed in the application programs

→ They are non-preemptive

→ Security is less compared to kernel level threads

→ When crashed it doesn't affect disrupt the hardware as it doesn't have direct access to the system calls.

Kernel level threads

→ These are threads of kernel process which are executed in kernel mode

→ Only Privileged processes takes place

→ They are pre-emptive

→ Security is higher than user level threads

→ But when executed & crashed it

spoils the entire hardware of the system but it is also complex to understand

→ It has better security than user level threads

5 a)

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
void main () {
    ptd_t ppid;
    ppid = fork();
    if (ppid == 0) {
        printf ("entered the inside of
                child process");
        exit(0);
    }
    else if(ppid < 0) {
        printf ("Error in the creation of
                child process");
        exit(0);
    } else {
        int status;
        pid_t pid;
        pid = wait (& status);
```

```
if (pid > 0) {
    if WIFEXECUTED(pid) {
        printf("child Executed Successfully");
    }
  }
 }
}
```

output:

```
Inside the child process
child Executed Successfully
```

In the above program the parent forks the child & waits for the completion of execution of the child process.

The wait() takes the status of the child & returns the PID of the child if it's less than 0 it has been terminated successfully.