190031527 Home Assignment-2 S. Krishna Vamsi

CO-2

1a) The U-area :- An operating system maintains a region called u-area i.e., user area which holds the specific information at process and stored in a stack segment.

* An array indicates how the process wishes to react to signals.
* An error feild records errors encountered during a system call
* A return value feild contains the result of System calls.

The process-Table :- The process table contains the information required by the kernel and the u-area holds the information required by the process itself

* process state
* process ID's
* Timer's for resourse usage

The xv6 kernel stack for a given process is located in the u-area for that process. A process u-area consists of the kernel stack and the user-structure.

The XVG kernel allocates one kernel stack per process retaining in the state of blocked processes on seperate kernel stacks in this way makes it easier to implement complex, high level functionality such as file systems within the kernel.

2a) A process run in cpu untill it is context switched. This happens when:

* The process exists
* The process uses up it's time slice.
* A resources has become available for sleeping process.

Sleep:- The algorithms for sleep, which changes the process state from "kernel running" to "asleep in the memory" and wakeup which changes the process from "asleep" to "ready to run".

Wakeup:- To wakeup sleeping process, the kernel executes the wakeup algorithm, either during the usual system call algorithms (or) when handling an interrupt.

For instances, the algorithm iput releases a locked inode and awakens all processes.

3) Algorithm for interrupt (Inthand):-

```
algorithm inthand /* handle interrupts */
input : none
output : none
{
   save(push) current context layer;
   determine interrupt source;
   find interrupt vectors;
    call interrupt handler;
}  restore (pop) previous context layer;
```

design of algorithm syscall:-

```
input : system call number
output :- result of system call.
{ find the entry in the system call table
   corresponding to system call number;
   determine number of parameters to
system call,
   invoke system call code in kernel;
if (error during execution of system call)
      { set register();
      }
else { set register 0,1;
      }
```

* small quantum reduces the response time for all processes, which is important to interact processes. However a long quantum reduces the overhead switching processes. A long quantum is useful for a batch system.

* I/O bound processes is favoured by a multi level feedback queuing schedular because jobs that use the cpu heavily will be moved to low priority queues

4)
```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#define size argc-1
int main (int argc , char **argv)
{ int userInput [size];
  int temp;
  int uisize = sizeof (userInput)
  int pipecom [2];
  pipe (pipecom);
pid_t pid = fork();
if (pid == 0)
{  int sum = 0;
   close (pipecom[1]);
  for (int i=0 ; i< uisize; i++)
   { read (pipecom[0] , &temp , sizeof (temp));
     sum += temp;
     temp = 0;
   }
```

```
        return sum;
    }
    else
    {   int sum = 0;
        for(int i = 1 ; i < argc ; i++)
        {
            userInput[i] = atoi(argv[i]);
        }
    int status;
    wait(&status);
        sum = WEXITSTATUS (status);
    printf ("Sum : %d \n", sum);
    return 0;
} }
```

5) Context Switch Flow:-

1) Interrupt the current process. Then, switch to cpu → scheduler.

2) Scheduler find a runnable proc in ptable and context switch from scheduler to Runnable Process.

process → scheduler → process

The scheduler is process which is a bridge between 2 processes.

After Context Switch:-

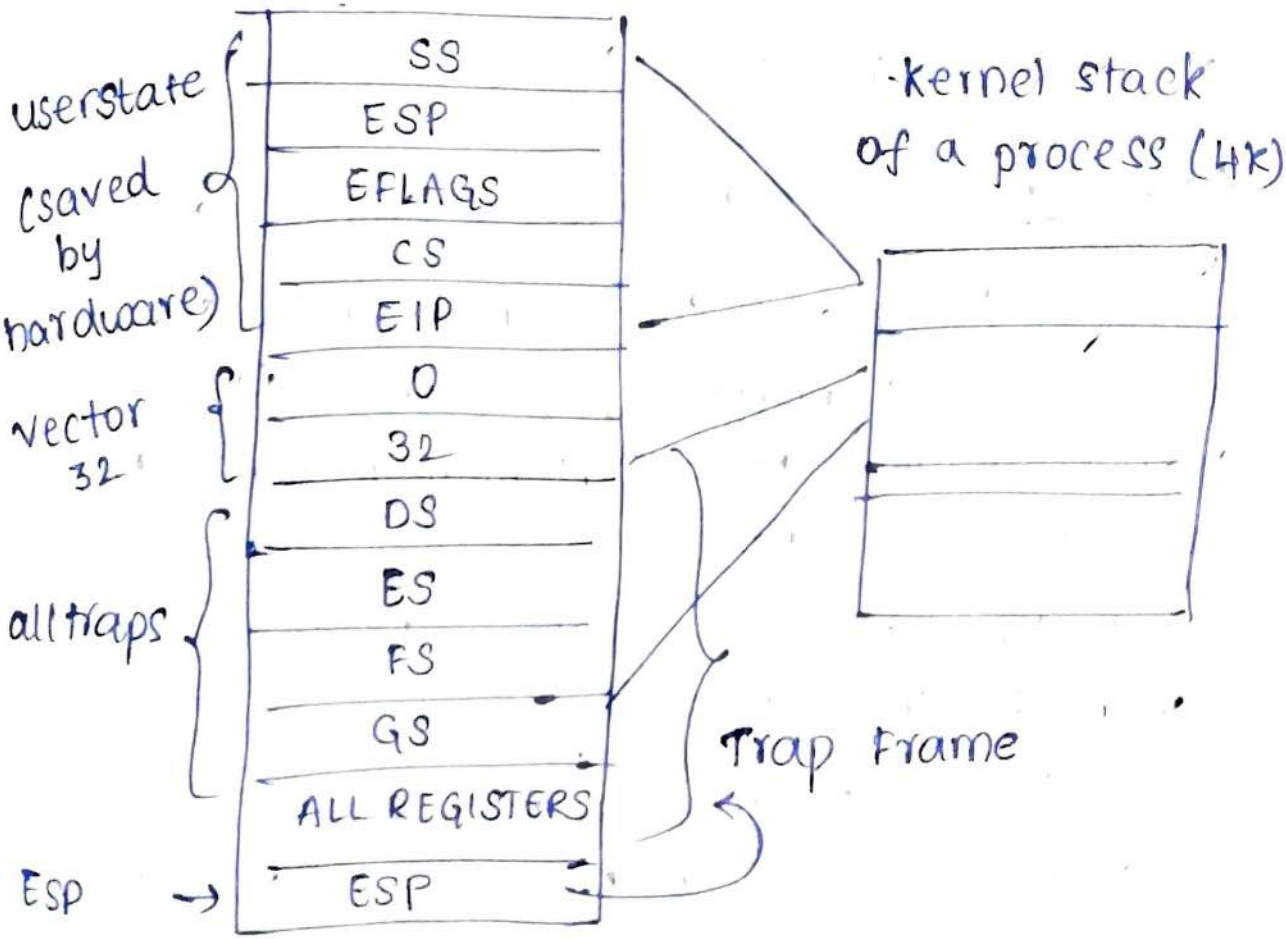
Destination : Exit back to user-level.

switch.S → Sched (proc.c) → yeild (proc.c)

→ trap (trap.c)

→ trapret (trapasm.S, but setup in allocproc())

→ iret (trapasm.S)



6) In computing a process is the instance of a computer program that is being executed by one (or) many threads it contains the program code and its activity depending upon the operating system, a process may be made up of multiple threads of execution of that execute instructions concurrently.

Tasks performed by fork() system call;

Systemcall fork() is used to create processes the purpose of fork() is to create a new process, which become the child process caller

* fork() returns a zero to the newly created child process.

*fork() returns a negetive value, the creation of child process unsuccesful.

7) SWITCH;

void swtch.(struct context **, struct context *);

* saves and restores contexts;
* context is a struct context *, stored on the kernel stack,
* cpu pushed onto stack. and saves stack pointer to *old.

SCEHD:- First we're acquiring a lock on the process table. This is to avoid race conditions when multiple processors are running in scheduling code. Then we set the current process state to RUNNABLE and we call a function called sched()

The majority of sched() is concenned with making sure it's safe to schedule a new process. First it checks that we're holding the process table lock.

8) (i) If a process occurs more than once in the round-robin list, then it will get a turns for each pass through the list. one reason for allowing this would be to implement a primitive priority system since the more times it occurs on the list

a) yes, because the scheduler alternates between the 2 queries, every job that is not executed from the "new" queue eventually ends up with old queue from there, eventually reaches the head of the queue and is executed.

No. It is not fair to all processes. Some lucky "new processes will get to execute when they reach the top of new queue, while some will drop to the

bottom of old queue and have to wait much longer for to execute these unlocked processes will have to wait for all the processes older than to complete and will have to wait for many processes younger than them to complete as well.

9) Every process in xv6 has a seperate area of memory called the kernel stack That is, allocated to it, to be used instead of the userspace stack when running in kernel mode.

The conceptual stages through which a process moves while being managed by the os are new, ready, running, waiting and terminated.

Kernel data structure: most of the kernel data structure are only accessible by the kernel data structure are only accessible by the kernel and it's subsystems. The may contain data as well as pointers to other data structures.

process :- process in the operating system can be in any one of states like new, ready, running etc.

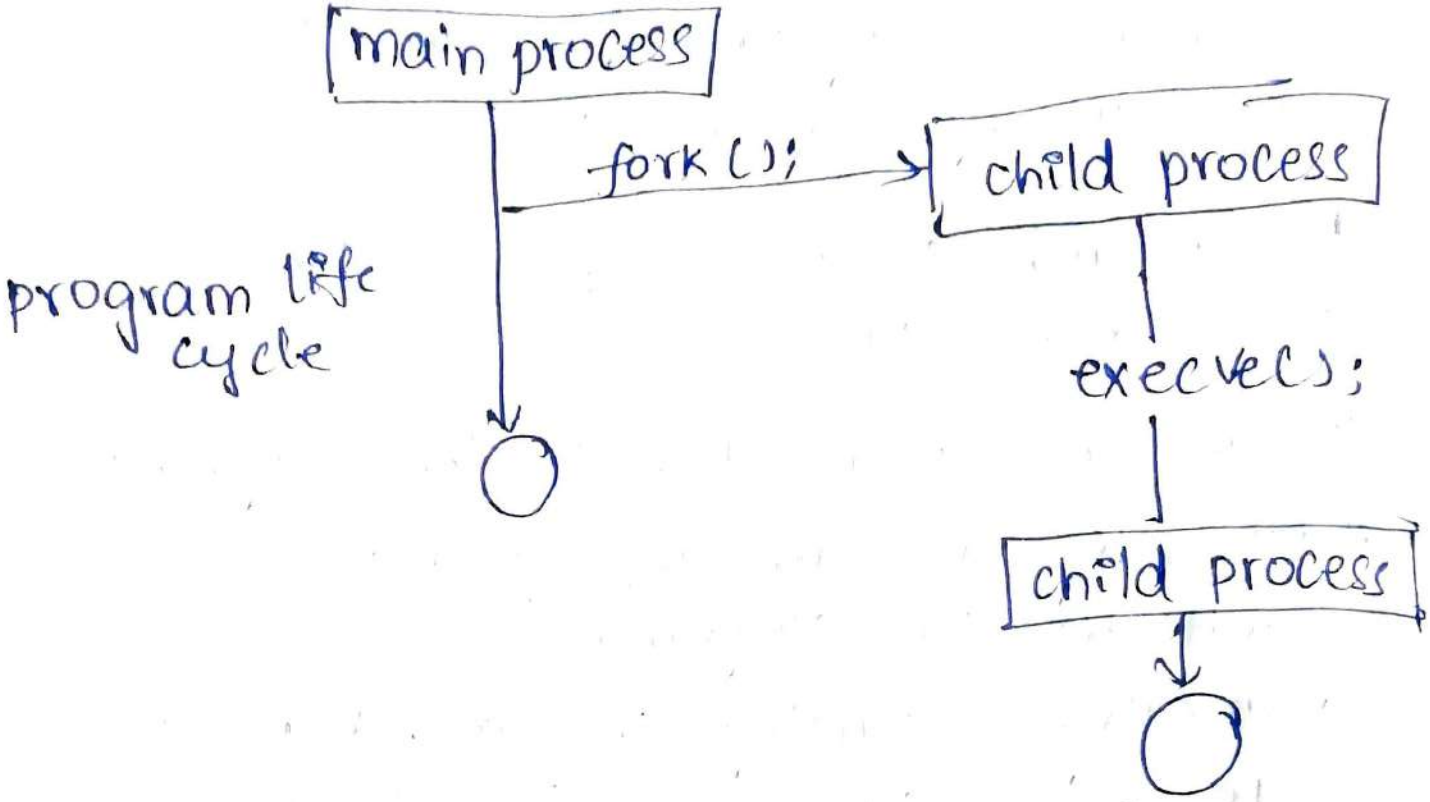
10) The normal unix boot process has these main phases

* Basic hardware detection (memory, disk, ...)
* Executing the firmware systems initialisation program.
* Locating & starting the unix kernel. The kernel image file to execute may be determined automatically or via input to boot program.
* The kernel starts init process, which inturn starts system processes and initialises all active subsystems when everything is ready, the system begins accepting user logins.

```
int execve(const char *file, char
        * const argv[]);
```

1) The first argument is the name of command.

2) The Second argument consists of the name of the command and arguments passed to command itself. It must also be terminated by NULL.



when the fork command completes, the child is an exact copy of parent process. However, when we invoke execve, it replaces the current program with the program passed into it in the arguments.