System Programming Notes

chap 2 (Unix System Programming) 8/10/23

Chap 2: Programs, Processes and Threads.

2.1

→ Process is an instance of program whose execution is started but not yet terminated.

→ A program is a prepared sequence of instructions to accomplish a defined task.

→ Steps of program to become a process:

program (P1.c) —compiler→ object file (P1.o)

P1.exe image ←loader— executable module (P1.exe) ←Linker

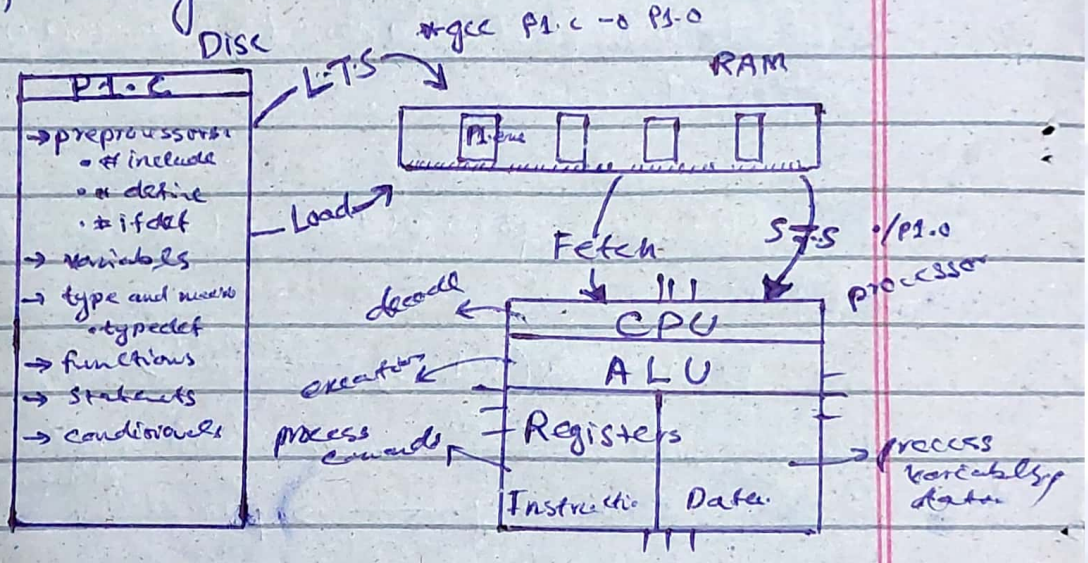
is loaded to memory → process is executed.

→ Each process has its own address, space, execution state, and process Id. OS manages these and many other resources for a process. It has also atleast one flow of control called thread.

→ Variables of a process can either be:
- static storage: remain in existance for life of process.

- automatic storage: allocated when execution enters a block and deallocated when leaves the block.

→ The program counter (PCB) keeps track of next instruction to be executed and is incremented after fetching an instruction.



2.2 PCB determines which instruction is next, The resulting stream of instructions, is called a Thread of Execution, can be represented by sequence of instruction address assign to PC during process execution.

→ Content switch is when execution switches from one to another process

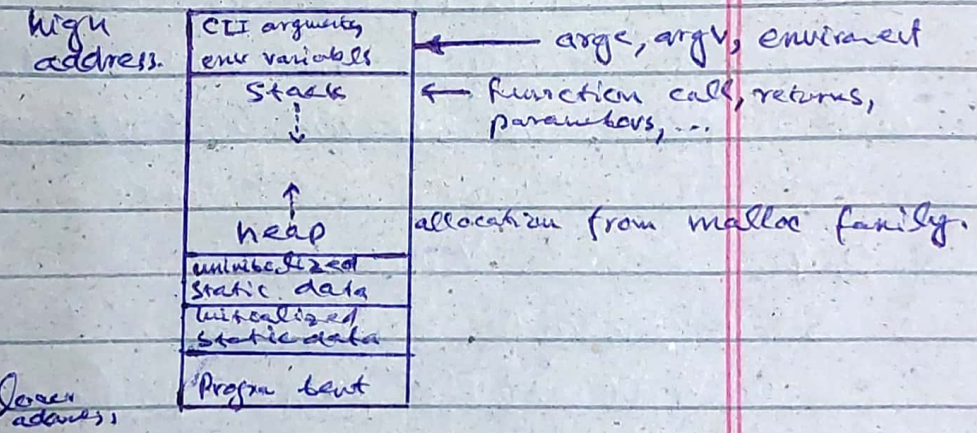
→ A thread is an abstract data type that represent a thread of execution within the process.

→ A thread has its own execution stack, program Counter value, register set, and state.

→ process are heavyweight while threads are lightweight processes.

2.3

→ After loading, the program executables appears to occupy a contiguous block of memory called program image.



→ Activation record is block of memory allocated on top of process stack. to hold execution content of a function during call. Each fuction

call creates a new activation record and removed when a function returns.

→ Each activation record contain: return address, parameters, status info, some registers values, automatic variables.

→ The malloc allocates from free memory pool called 'heap'. storge on a heap persist until it is freed by program are when it exits.

→ The program image appears to occupy a contiguous memory but in practice the OS maps it to non-contiguous blocks of physical memory called pages.

2.4

→
```
#include <unistd.h>
int close (int filedes);
```
→ If succesful close returns 0.
→ If unsuccessful close returns -1 and also set errno.

→ #include <stdio.h>
void perror(const char *s);
- No return values and no errors are defined.
- Output to standard error a message.

→ #include <string.h>
char *strerror(int errnum);

- if successful strerror return a pointer to error string. No values are reserved for failure.
- Use to produce informative message of corresponding errnos.
- This fuction may change errno, save and restore if needed again.

2.5 Functions Return values and Errors:

→ functions should never exit on their own, but rather should ~~exit~~ always indicate an error to the calling program.

→ Usually, a function that allocates memory should either free the memory or make a pointer available to the calling program.

→ Failure of library function doesn't cause our program to stop.

→ Our custom function should handle error using this approach:
- Print error messag in main only.
- return -1 or Null and set errno indicator
- return an errorcode.

2.8 Use of Static Variables

→ Care must be taken in using static variables. in situations with multiple threads, static variables are useful.

2.10 Process Environment:

→ Environment list consists of an array of pointers to string of the form 'name = value'. The name specify an environ variable and the value specifies a string value associate with environment variable. The last entry of the array is NULL.

→ The external variable 'environ' points to process environment list when the process begins executing.

SYNOPSIS:

```
extern char **environ;
```

→ If process is executed by 'execl', 'execlp', 'execv' or 'execvp', the process inherit the env list from its parent.

→ `char *getenv(const char *name);`

- returns NULL if name doesn't have a value.
- returns pointer to string if name has a value.

2.11 Process Termination:

→ When process terminates, the OS deallocates the process resources, updates the appropriate statastics and notify other process.

→ Termination can be normal or abnormal.

→ normal termination occur under the following conditions.

- return from main
- Implicit return from main
- call to exit, _Exit or _exit

→ The exit function call user-define exit handlers that were registered by 'atexit' in reverse order of registration.

→ The _exit or _Exit do not call user-defined handlers.

→ SYNOPSIS

```
#include <stdlib.h>
void exit (int status);
void _Exit (int status);

#include <unistd.h>
void _exit (int status);
```

→ The 'atexit' functions installs a user-define exit handler. Exit handlers are executed on a last-installed-first-executed order when program terminates by return or exit call.

```
#include <stdlib.h>
int atexit (void (*func)(void));
```

- If successful atexit returns 0
- If unsuccesful atexit returns a nonzero value.

→ A process can also terminates abnormally by calling 'abort'

or by passing a signal (like ctrl + C).