28 Sept, 2023

Systems Programming

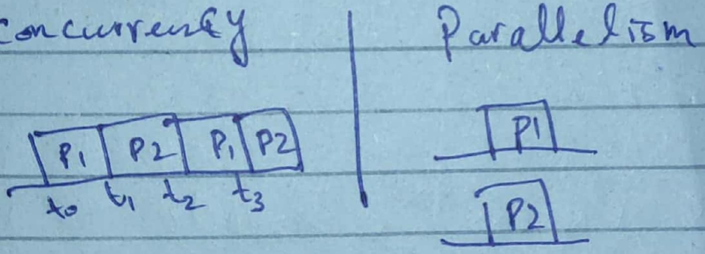
Book: Unix Systems Programming: Communication Concurreny & Threads (Author K. A. Robins & Steven Robins)

Contents:

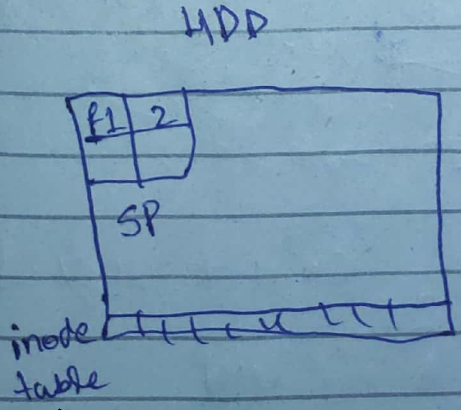
Midterm
1. ____ self study → Reading Assignment
2. Program, Process, threads
3. Processes
4. Unix I/O (standard I/O device) (Regular files)

represent memory Address of func
main() {
Scanf(); → sys call

5.1
5. Files & Directories
6. Special files (fifos / pipes / sockets):
8. Signals



9. Times & Timers
13. Threads / 15. IPC using shared Memory.



Signal:- notification of event.

By ALi Asghar

Limitless Games Studio

M. term

CLO's :-

CLO 1: error handling / file handling / Signal handling.

F.Term

CLO2: parallel processing (multiprocessing / multithreading

F.term CLO3: Interprocess Communication for real world.

Grading Criteria :-

Mid : 30%
final : 40%
Sessional : 30%
Project / Attendance / Quizzes / Assignment
3-5 3

Oct 5, 2023

Programs, Process & Threads

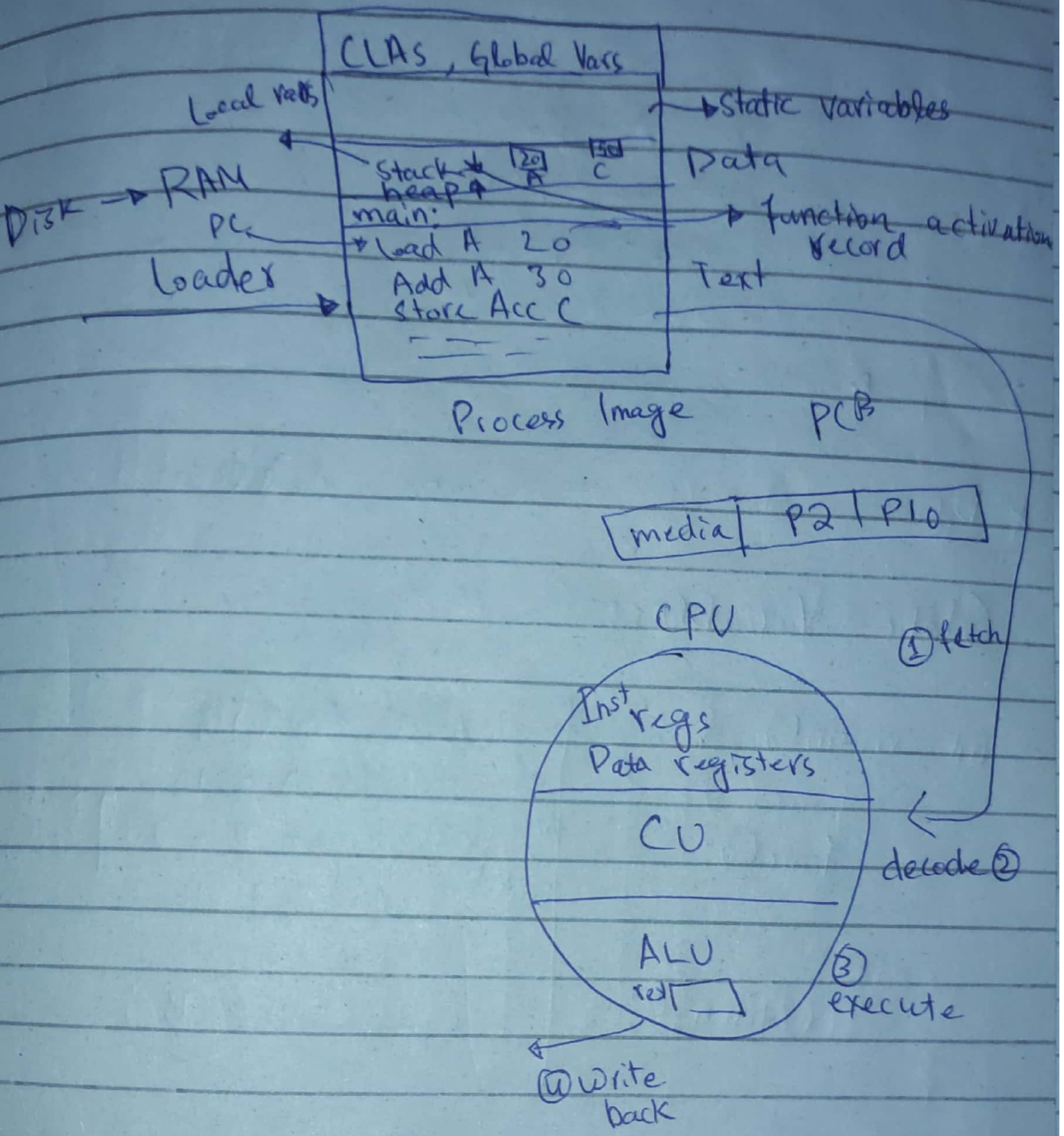
Program → Process

```
#include <____>      } → pre-compiler directives
#define COUNT 30     }   → do not compile
```

pre compiler, preprocessor directives

```
    starting point
int main(int argc, char * argv[]){
        int A = 20;
        int C = A + COUNT
        printf("%d \n", C);
        return 0; → exit status returned to pare
}
```

gcc $\underbrace{\text{p1.c}}_{\text{disk}}$ -o $\underbrace{\text{p1.o}}_{\text{disk}}$



Variables
- scope
- usage
- lifetime

```
int main ( ) {
        int A = 20;  int c = A + COUNT;
        fun(A);  ——▸ stack will be
                       divided
```

fun and main will have different stacks

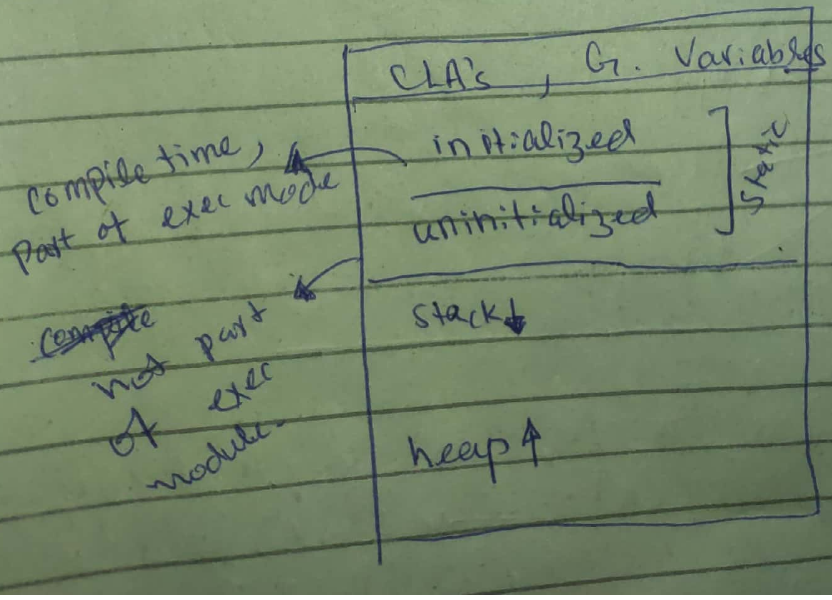
function stack will be destroyed whe
it returns.

```
fun(int A){
        int X = 30;
        int y = 40;
        X++; y++;
        cout<< X<<Y;
   }
```

★ Dynamic Memory must be deallocated

Static Variables:

```
int main() {                 void fun(){
     func();                      int A = 10;
     func();             dec    Static int B = 20;   execute
                          +                          only once
                         init     A++;
                                  B++;
     return 0;                    cout << A << B ;
}
```



Kernel manage process in the form of
PCB → A linked list.

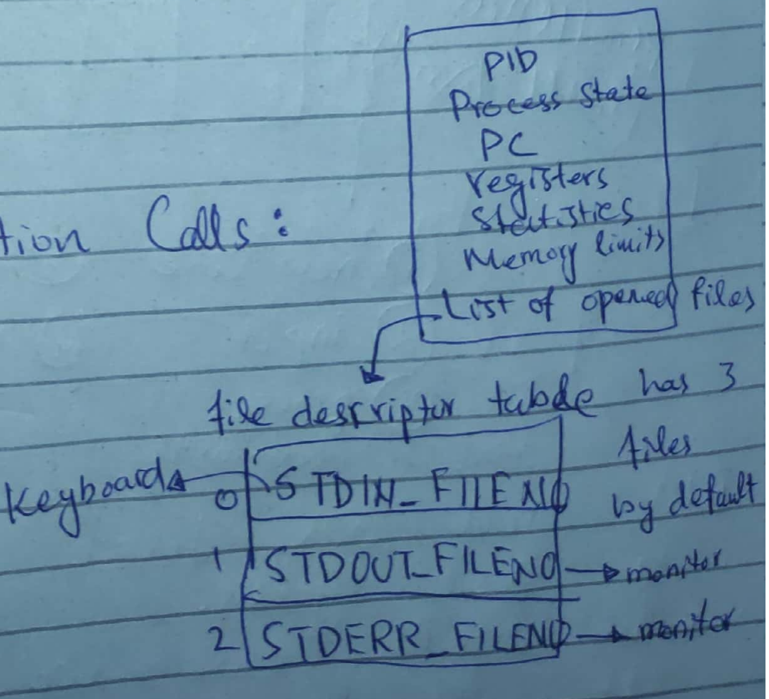
Locals vars are destroyed while static aren't.

```
void fun() {
        static int c;
            c = 10;
            c++;
          cout << c;
```

Static int A [100];	static int A [100] = {1, 2, 3};
400 bytes	400 + 100 * 4 + 7 = 807

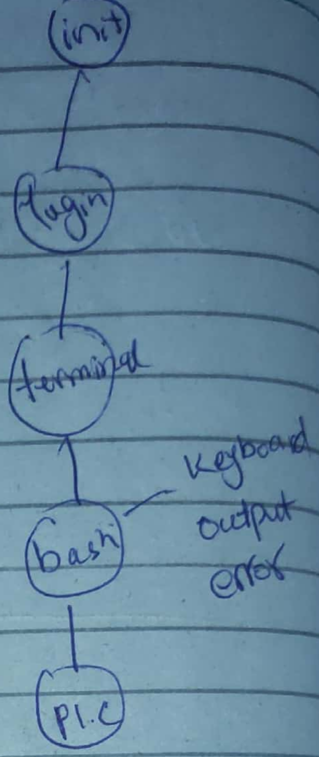
Oct 6, 2023

Library Function Calls:



PID
Process State
PC
registers
statistics
Memory limits
List of opened files
file descriptor table has 3 files by default
Keyboard
0 STDIN_FILENO
1 STDOUT_FILENO → monitor
2 STDERR_FILENO → monitor

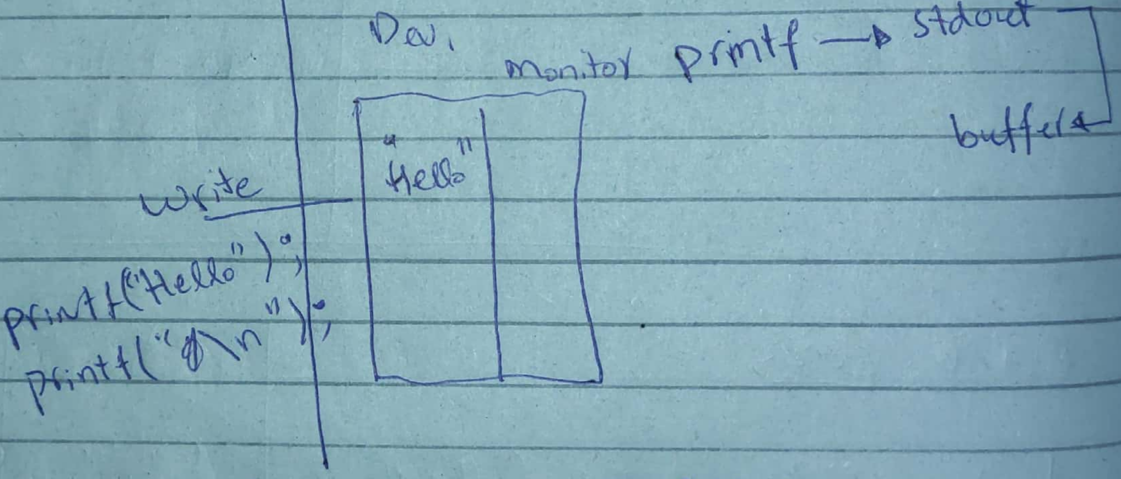
→* Child process will have Same PC, registers and list of opened files.



Stdin, stdout, stderr are opened by defult.

* why two entries for monitor.

Stdout | vs Stderror



① new line
② input
⑤ process terminate
③ fflush
④ buffer full

→ We deal with device buffer in order to Save cycles.

→ In errors case, we want to let user know immediately. So stderror has no buffer.

→ buffer is memory in RAM.

```
main() {
        printf("Hello1"); ——→ stdout
        ~~Printf~~ ("Hello2"); ——→ Stderr
        perror
}
  Output:   Hello2 Success
            Hello1
```

```
                 filename     string
fprintf( stdout,  "Hello1");
fprintf ( Stderr, "Hello2");
```

Library function Calls :-

① Detect error
② Error Reason

```
return_type   name (args);
    int

    int  close( int fd);
  -1 : error
errno ←
   0 : success
```

0	Std in
1	Std out
2	error

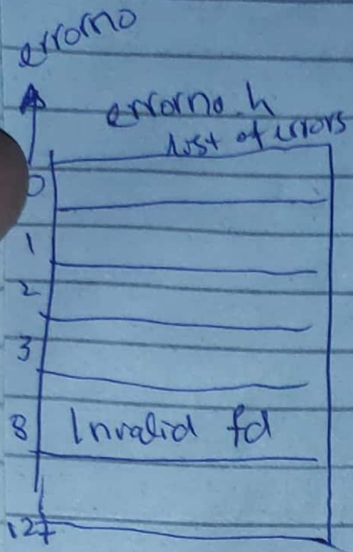
```
#include<errno.h>
#include <string.h>
```

int errno

② set errno

-1 ①Return failure

```
int main()
    int ret = close(4);
    if (ret == -1){ // error
perror("error"); printf("Error = %d \n", errno
        fprintf(stderr, "Failed to clo
                 : %s", stderr(errorno)
        return -1
    }
```



```
char* strerror(int errno);
```

close(1) → will close stdout and printf will fail.

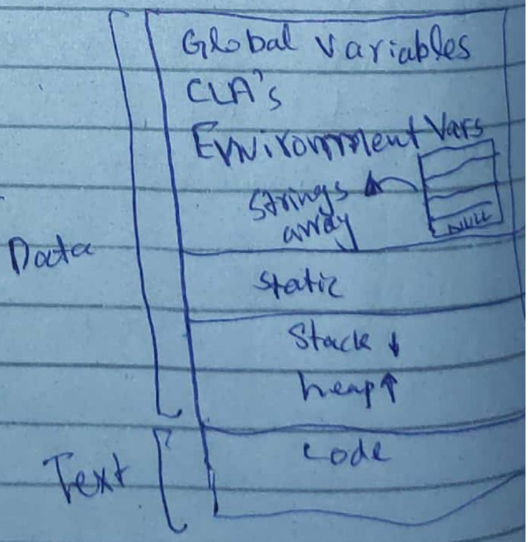
Date 9 Oct, 2023

Process Environment

—— $env

↳ List Environment Vars

ls → pwd
↓
PATH



$\underbrace{\text{HOME}}_{\text{Var name}} = \underbrace{\text{/home/student}}_{\text{Value}}$

```
echo $HOME
```

Environment vars are also inherited by child process.

```
extern char ** environ
```

↳ Let compiler know that we are using it from another source file.
→ it will Not be declared as a new var.

```
main() {
      for( int i = 0; environ[i] != NULL; i++)
            printf("% \n", environ[i]);
}
```

```
Char* getenv(const char * varname);
```

```
main(){
      char * valofpath;
      valofpath = getenv("PATH");
      printf ("PATH= %s\n", valofpath);
```

Normal
Process Termination:

(stty) Exit handlers

same

1. Explicit $\boxed{\text{exit}}$ → different
2. Implicit exit (return statement of main)

3. $\boxed{\text{\_exit() / \_Exit()}}$ same

```
int atexit( void * fun (void))
```
0: success
non-zero: failure

```
int main ( ) {
        atexit(fun1); atexit(fun2);
  ①    printf("In main \n");
        exit(0);
}
```

```
void
fun1( void) {
② printf("Process
  Terminating Normally
  \n");
}

void fun2(void){
③ printf(" All s
  executed\n");
}
```

```
_exit(0); // does not call
          // exit handlers
```

Scanned with CamScanner

```
int main () {
    atexit( fun1);        atexit(fun2);
    int ret = close(4);
    if ( ret == -1) {
        fprintf(stderr, "Failed close sc");
        _exit(-1);
    }
    return 0;
}
```

→ String Tokenization Skipped.

→ Chap 2 Quiz in next class.