**Day-22**

**System programming**

**Kernel mode vs user mode**

* **Kernel mode:**
* The CPU operates in a privileged mode where it has unrestricted access to hardware resources.
* It can execute all instructions and access protected areas of memory.
* **user mode**
* restricted mode, limited accessing specific memory areas and cannot directly interact with hardware.
* If an application need access, it must request the kernel via **system calls.**
* Example: In C,
* **Why system calls?**
* system calls act as an interface between user program and the operating system.
* They enable user-mode application to request kernel services like file handling, process management, network communication.
* **Need of system calls:**
* Direct HW access by user programs can lead to system instability.
* To read mac address, encryption, decryption.
* Example: in C functions like open(), read(),and write() are wrappers for system calls.
* A **MAC address** is a **unique identifier** assigned to network interfaces for communication at the data link layer (Layer 2) of the OSI model. It is used in various networking technologies, including Ethernet, Wi-Fi, and Bluetooth, to identify devices on a local network.
* **System call execution and types:**
* **Execution process:**
* A user mode program issues a system call (e.g. read())
* The program switches from user mode to kernel mode.

**Process control**: creation, termination, and management of processes.

Ex: fork (), exec(), exit()

**File management**: operation like reading, writing and closing, opening

**Device management**: interacting with the HW devices.

Ex: ioctl(), read()

**Information maintenance**: accessing system information

Ex: getpid(), gettimeofday()

**Communication**: managing inter-process communication

Ex: pipe (), socket ()

Example:

#include<fcntl.h>

#include<

* **Command to view system calls - strace**

**Purpose:**

* Strace is a debugging tool in Linux used to trace system calls made by a program.
* It shows all interactions between the program and the operating system kernel.
* **Usage :**

strace ./your\_program

**………………………………………………………………………………………**

* user69@trainux01:~$ strace ./a1.c
* execve("./a1.c", ["./a1.c"], 0x7ffe81cbf9f0 /\* 21 vars \*/) = -1 EACCES (Permissi on denied)
* fstat(2, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 25), ...}) = 0
* write(2, "strace: exec: Permission denied\n", 32strace: exec: Permission denied
* ) = 32
* getpid() = 1560
* exit\_group(1) = ?
* +++ exited with 1 +++
* user69@trainux01:~$

**………………………………………………………………………………………………**

**Process Basics**

1. **process control block (PCB):**

* It is a data structure maintained by the operating system to store all information about specific process.
* It acts as a repository for a process’s attributes and is essential for a process management.

**Contents of PCB:**

* Process ID(PID)
* Process state
* Program counter
* Registers
* Memory management information
* Accounting information
* I/O information
* Scheduling information

**Role:**

* Critical for content switching process, as it saves and restores the state of process

1. **Process, PCB, Attributes, state and scheduling**

**Process:**

1.

2. Parent process

3. Priority

4. Execution content

5. Resource usage

**Process state:**

1. New
2. Ready
3. Running
4. Waiting
5. Terminated

**Scheduling:**

* It determines which process gets the CPU and for how long.
* It is managed by the scheduler in the OS.
* Types:

1. Long term
2. Medium
3. short term

* **Context testing**

1. Save current process
2. Update
3. Restore

………………………………………………………………………………

* While using system programming “defunct” should not be present.

………………………………………………………………………………

* getpid() returns process ID
* getsspid()

………………………………………………………………………………

#include <stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

int main() {

long int i;

int pid = getpid();

pid\_t ppid =getppid();

printf("\nMY OWN Pid=%d\n",pid);

printf("\nfor %d parent is %d\n",pid,ppid);

printf("\n\n");

return 0;

}

Output:

MY OWN Pid=18580

for 18580 parent is 13078

……………………………………………………………………………

ps -ef

……………………………………………………………………………

**Fork:**

* create a child process

SYNOPSIS

#include <sys/types.h>

#include <unistd.h>

pid\_t fork(void);

* fork () creates a new process by duplicating the calling process.
* The new process is referred to as the child process.
* The calling process is referred to as the parent process.
* The child process and the parent process run in separate memory spaces.
* Memory writes, file mappings (mmap(2)), and unmappings (munmap(2)) performed by one of the processes do not affect the other.

……………………………………………………………………………….

* Fork is returning the child’s pid in the parents’ process
* Fork is returning 0 in child’s process.

…………………………………………………………………………….

|  |
| --- |
| #include <stdio.h>  #include<stdlib.h>  #include<sys/types.h>  #include<unistd.h>  int main() {  printf("\nI am parent and my id is %d\n",getpid());  printf("\nparents ppid: %d",getpid());  pid\_t id=fork();  printf("\nHello World\n");  printf("\npid: %d",getpid());  printf("\n\n");  return 0;  }  **Output:**  I am parent and my id is 24466  parents ppid: 24466  Hello World  pid: 24466  parents ppid: 24466  Hello World  pid: 24467 |

|  |
| --- |
| #include <stdio.h>  #include<stdlib.h>  #include<sys/types.h>  #include<unistd.h>  int main() {  printf("\nI am parent and my id is %d\n",getpid());  printf("\nparents ppid: %d",getpid());  pid\_t id=fork();  printf("\nHello World\n");  printf("\npid: %d",getpid());  printf("\nFork returned id =%d",id);  printf("\n\n");  return 0;  }  Ouput :  I am parent and my id is 24755  parents ppid: 24755  Hello World  pid: 24755  Fork returned id =24756  parents ppid: 24755  Hello World  pid: 24756  Fork returned id =0 |

|  |
| --- |
| #include <stdio.h>  #include<stdlib.h>  #include<sys/types.h>  #include<unistd.h>  int main() {  printf("\nI am parent and my id is %d\n",getpid());  printf("\nparents ppid: %d",getpid());  pid\_t id=fork();  if(id==0)  {  printf("\nI am child process\n");  printf("\nmy id=%d",getpid());  }  else  {  printf("\nIn the parent process id=%d ",id);  }  printf("\nHello World\n");  printf("\n\n");  return 0;  }  **Output:**  I am parent and my id is 25070  parents ppid: 25070:  In the parent process id=25071  Hello World  parents ppid: 25070  I am child process  my id=25071  Hello World |

|  |
| --- |
| #include <stdio.h>  #include<stdlib.h>  #include<sys/types.h>  #include<unistd.h>  int main() {  fork();  fork();  printf("\nHello World\n");  printf("\n\n");  return 0;  }  **Output:**  Hello World  Hello World  Hello World  Hello World |

* If we use 3 fork () it will print 8-times

|  |
| --- |
| #include <stdio.h>  #include<stdlib.h>  #include<sys/types.h>  #include<unistd.h>  int main() {  int a=10;  int pid;  pid=fork();  if(pid<0)  {  printf("\nunable to fork()");  exit(EXIT\_FAILURE);  }  else if(pid==0)  {  a+=20;  printf("\nchild a=%d\n",a);  }  else  {  a\*=2;  printf("\nparent a=%d\n",a);  }  printf("\nA=%d\n",a);  return 0;  }  Output:  parent a=20  A=20  child a=30  A=30 |

**WAIT:**

* wait, waitpid, waitid - wait for process to change state.