# OPERATING SYSTEM ASSIGNMENT

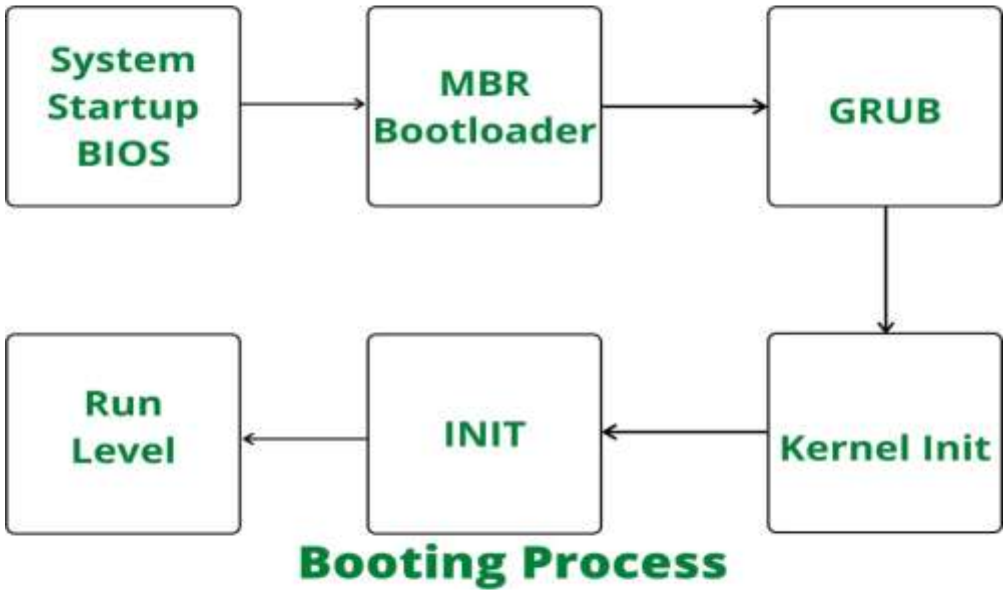
Q1. Write the difference between l1 ,l2 and l3 cache memory.

| L1 cache memory | L2 cache memory | L3 cache memory |
|---|---|---|
| It is directly built into the processor and it is used to store CPU's recently accessed information | It is located outside and separated from the CPU chip core. | It is used by CPU and it is built on the motherboard within CPU module itself. |
| Smallest cache | Larger than L1 and smaller than L3 | Largest cache |
| Called level 1 or primary or internal cache | Called level 2 or secondary or external cache | Called level 3 or external cache |
| Fastest cache | Slower than L1 but faster than L3 | Slowest cache |
| Each core in CPU has their own L1 cache memory | Each core in the CPU has their own L2 cache memory | All cores in the CPU share the same L3 cache memory |

Q2. Explain the process of booting and virtual memory

in linux. Ans.= ***Stages of Linux Boot Process**

1. The machine's BIOS or boot microcode hundreds and runs a boot loader.
2. Boot loader finds the kernel image on the disk and loads it into memory, to start the system.
3. The kernel initializes the devices and their drivers.
4. The kernel mounts the basis filesystem.
5. The kernel starts a program referred to as init with a method ID zero
6. init sets the remainder of the system processes in motion.
7. For some purpose, init starts a method permitting you to log in, typically at the top or close to the top of the boot sequence.



- **Virtual memory in linux**

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.

The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory is available not by the actual number of the main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

## Q3. Write the difference between MBR and GPT.

Ans.= **Master Boot Record** (MBR) and **GUID Partition Table** (GPT) are two partitioning schemes for hard drives everywhere, with GPT being the newer standard. For each option, the boot structure and the way data gets handled are unique. Speed varies between the two partition options, and requirements are also different.

**MBR** manages how partitions are created and organized on the **hard disk drive** (HDD). MBR uses Bios firmware and stores code in the disk’s first sector with a **logical block address (LBA)** of 1. The data includes information related to how and where Windows resides to manage the boot process in the PC’s primary storage and internal random access memory (RAM), not external memory such as DDR2 and DDR3 memory cards/sticks.

**GPT** stands for **G**UID **P**artition **T**able. Just like MBR, it also manages the creation and organization of partitions on the HDD. GPT uses UEFI firmware, and it also stores disk information, such as partitions, sizes, and other essential data, just like MBR does in sector one. However, GPT uses sector two because sector one gets reserved for MBR and BIOS compatibility. In GPT technical terms, MBR sector #1 (LBA 1) is LBA 0 for GPT, and GPT is sector 1 (LBA 1).s

Q4. What are the main differences between processes and threads.

Ans.=

| PROCESS | THREAD |
|---|---|
| Process means any program is in execution. | Thread means a segment of a process. |
| The process takes more time to terminate. | The thread takes less time to terminate |
| It takes more time for creation. | It takes less time for creation. |
| It also takes more time for context switching | It takes less time for context switching. |
| The process is less efficient in terms of communication. | Thread is more efficient in terms of communication. |
| Multiprogramming holds the concepts of multi-process. | We don't need multi programs in action for multiple threads because a single process consists of multiple threads. |
| The process is isolated. | Threads share memory. |
| If one process is blocked then it will not affect the execution of other processes | If a user-level thread is blocked, then all other user-level threads are blocked. |
| The process has its own Process Control Block, Stack, and Address Space. | Thread has Parents' PCB, its own Thread Control Block, and Stack and common Address space. |
| The process is called the heavyweight process. | A Thread is lightweight as each thread in a process shares code, data, and resources. |

Q5. What is critical section problem and the three conditions for its solutions.

Ans.=

- **Critical Section Problem**

Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.

```
do{
entry section
    critical section
exit section
     remainder section
}while(TRUE);
```

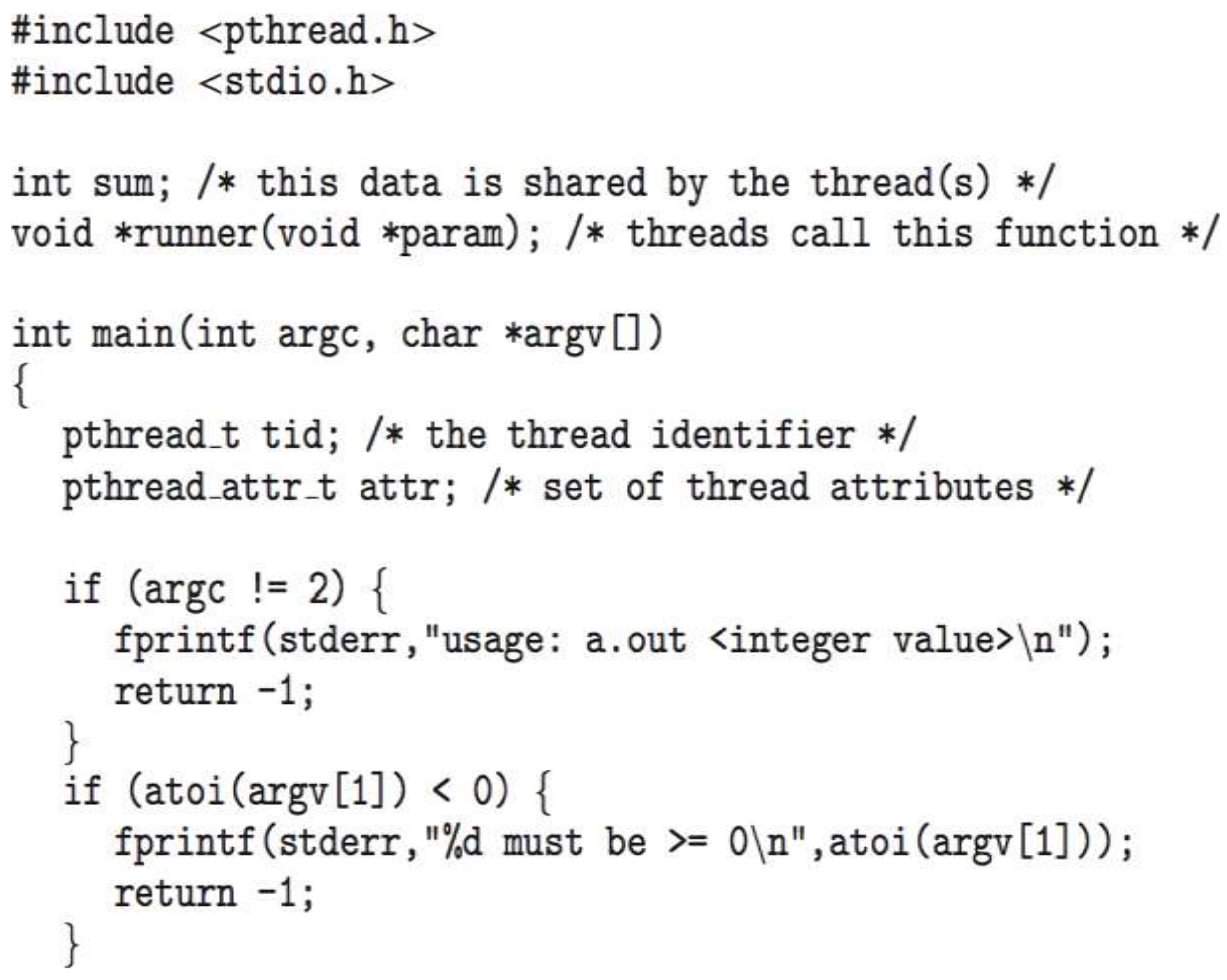
In the entry section, the process requests for entry in the Critical Section.

Any solution to the critical section problem must satisfy three requirements:

- **Mutual Exclusion** : If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress** : If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, and the selection can not be postponed indefinitely.
- **Bounded Waiting** : A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

**CODES:**

- <u>Pthread sum of 2 non negative threads.</u>

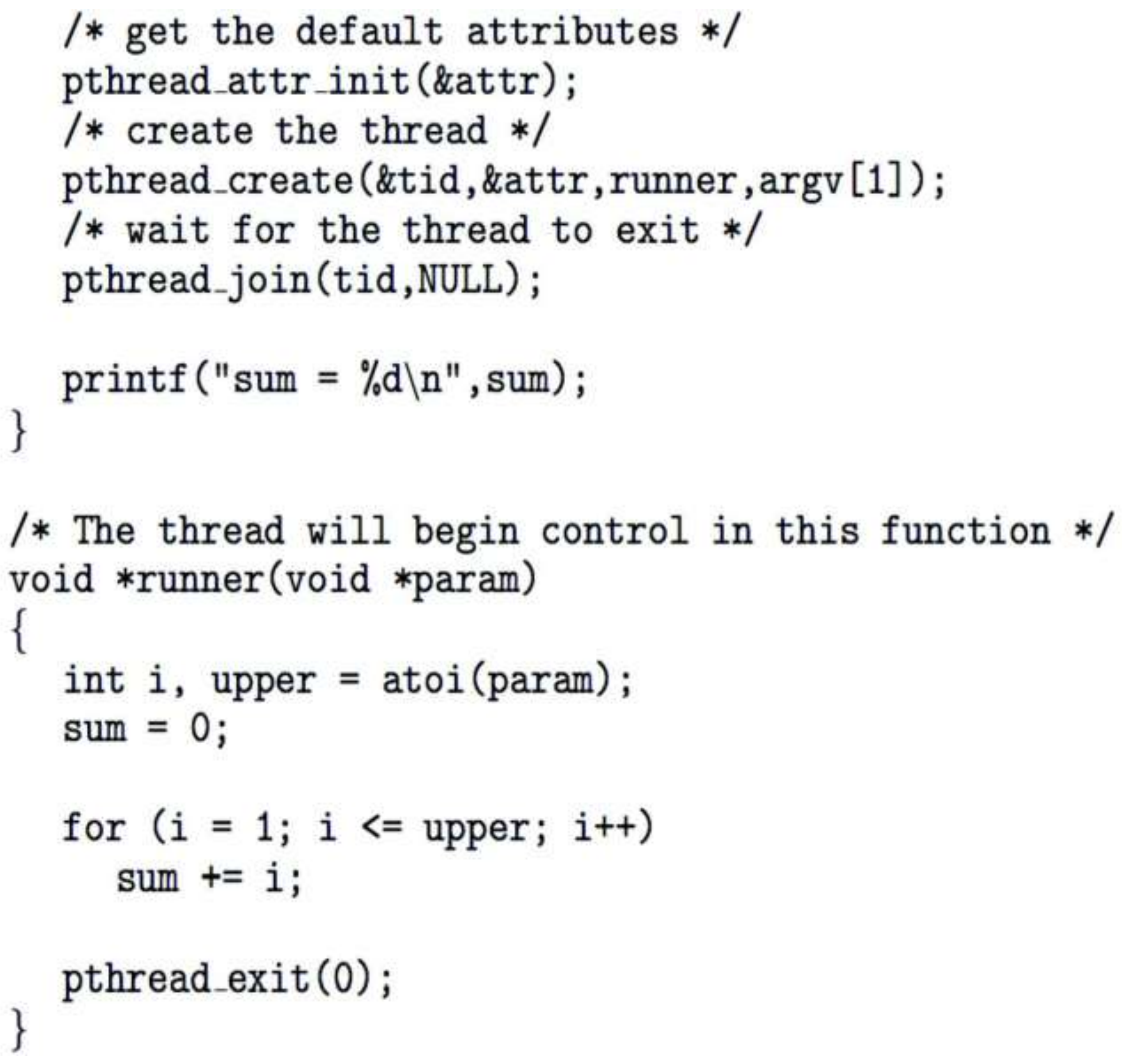


```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
  }
  if (atoi(argv[1]) < 0) {
    fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
    return -1;
  }
```



```
  /* get the default attributes */
  pthread_attr_init(&attr);
  /* create the thread */
  pthread_create(&tid,&attr,runner,argv[1]);
  /* wait for the thread to exit */
  pthread_join(tid,NULL);

  printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
  int i, upper = atoi(param);
  sum = 0;

  for (i = 1; i <= upper; i++)
    sum += i;

  pthread_exit(0);
}
```

- **Pthread Creation**

```
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS4
```

```
void hello (void *arg){/ thread main */
    printf("Hello Thread\n");
    return 0;
int main (void){
    int i;
    pthread_t tid[NUM_THREADS);
    for (i=0;i<NUM_THREADS; i++){/* create/fork threads */
       pthread create(&tid [i], NULL, hello, NUL):
    for (i=0;i<NUM_THREADS; i++){/* wait/jo reads */
        pthread_join(tid[i], NULL);
    return 0;
```

- **POSIX scheduling**

```
#include <pthread.h>
 #include <stdio.h>
#define NUM THREADS 5
int main(int argc, char * argv[])

int i, policy;

pthread t tid[NUM THREADS];
pthread attr t attr;

/* get the default attributes */
 pthread attr init(&attr);
```

```
/* get the current scheduling policy */
if (pthread attr getschedpolicy(&attr, &policy) != 0
fprintf(stderr, "Unable to get policy. backslash n^ prime prime )
else {


if (policy == SCHED OTHER)
 printf("SCHED OTHER backslash n^ prime prime );
 else if (policy== SCHED RR) printf("SCHED RR backslash n^
prime prime );
 else if (policy == SCHED FIFO)
printf("SCHED FIFO backslash n^ prime prime rangle; }


/* set the scheduling policy - FIFO, RR, or OTHER */
if (pthread attr setschedpolicy(&attr, SCHED FIFO) != 0)
fprintf(stderr, "Unable to set policy. backslash n^ prime prime )


/* create the threads */
for (i = 0; i <NUM THREADS; i++)
pthread create(&tid[i],&attr,runner,NULL);
/* now join on each thread */
for (i = 0; i < NUM THREADS; i++)
pthread join(tid[i], NULL);


}
/* Each thread will begin control in this function */ void *runner(void
*param)
{
/* do some work... */
pthread exit(0);


}
```

- **Fork function**

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int value = 5;
int main() {
pid t pid;
pid = fork();
if(pid == 0) { /* Child Process */
value= value + 15;
printf("CHILD PROCESS TERMINATED \n");
return 0;
}
else if(pid > 0) { /* Parent Process */
wait(NULL); printf("PARENT: value=%d", value);
printf("\n"); printf("PARENT PROCESS TERMINATED \n");
return 0;
}
else
printf("Fork failed");
return 0;
}
```

Output = 5….