Institute of Engineering & Technology

Devi Ahilya Vishwavidyalaya, Indore (M.P)

Department of Computer Science & Engineering

Operating System (CER4C2)

Lab-Assignments

Submitted To:

Er. Lalit Gehlod Sir

CS-Dept

IET-DAVV

Submitted By:

Tanishq Chauhan (21C4184)

CS "B" 2nd Year

Lab Assignments

1. Write the difference between L1,L2,L3 cache.

L1 CACHE	L2 CACHE	L3 CACHE
A cache memory that is directly built into the processor and is used to store the CPU's recently accessed information.	A cache memory that is located outside and separated from the CPU chip core, although it is found on the same CPU chip package.	A cache memory that is used by the CPU and is usually built onto the motherboard within the CPU module itself.
L1 cache is the smallest cache.	Larger than L1 but smaller than L3.	L3 cache is the largest cache.
L1 cache is called level 1 or primary or internal cache.	L2 cache is called level 2 or secondary or external cache.	L3 cache is called level 3 or external cache.
L1 cache is the fastest cache.	Slower than L1 but faster than L3.	L3 cache is the slowest cache.
Each core in the CPU has their own L1 cache memory.	Each core in the CPU has their own L2 cache.	All cores in the CPU share the same L3 cache memory.

2. Explain the process of booting and virtual memory in Linux.

I. PC On

II. CPU initializes itself and looks for a firmware program (BIOS) stored in BIOS Chip **(**Basic input-output system chip is a ROM chip found on mother board that allows to access & setup computer system at most basic level.)
In modern PCs, CPU loads UEFI (Unified extensible firmware interface)

III. **CPU** runs the BIOS which tests and initializes system hardware. BIOS loads configuration settings. If something is not appropriate (like missing RAM) error is thrown and boot process is stopped.
This is called **POST** (Power on self-test) process.
(UEFI can do a lot more than just initialize hardware; it's really a tiny operating system. For example, Intel CPUs have the Intel Management Engine. This provides a variety of features, including powering Intel's Active Management Technology, which allows for remote management of business PCs.)

IV. **BIOS** will handoff responsibility for booting your PC to your OS's bootloader.
1. BIOS looked at the MBR (master boot record), a special boot sector at the beginning of a disk. The MBR contains code that loads the rest of the operating system, known as a "bootloader."

The BIOS executes the bootloader, which takes it from there and begins booting the actual operating system—Windows or Linux, for example.
In other words,
the BIOS or UEFI examines a storage device on your system to look for a small program, either in the MBR or on an EFI system partition, and runs it.

V. The bootloader is a small program that has the large task of booting the rest of the operating system (Boots Kernel then, User Space). Windows uses a bootloader named Windows Boot Manager (Bootmgr.exe), most Linux systems use GRUB, and Macs use something called boot.efi

Virtual Memory Implementation

Linux supports virtual memory, that is, using a disk as an extension of RAM so that the effective size of usable memory grows correspondingly. The kernel will write the contents of a currently unused block of memory to the hard disk so that the memory can be used for another purpose.

Implementation

Virtual memory is implemented using Demand Paging or Demand Segmentation. Demand Paging : The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging. … The required page will be brought from logical address space to physical address space.

3. What is the difference between MBR and GPT.

MBR	GPT
Supports up to 4 primary partitions.	Supports up to 128 primary partitions.
The maximum capacity of MBR partition tables is only about 2 terabytes.	GPT partition tables offer a maximum capacity of 9.7 zetabytes.
Associated with BIOS.	Associated with UEFI.
MBR does not store copy of information.	GPT stores a copy of information.
If any corruption occurs in the information then the data can not be recovered.	If any corruption occurs in the information then the data can be recovered.

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

Comparison Basis	Process	Thread
Definition	A process is a program under execution i.e an active program.	A thread is a lightweight process that can be managed independently by a scheduler.
Context switching time	Processes require more time for context switching as they are more heavy.	Threads require less time for context switching as they are lighter than processes.
Memory Sharing	Processes are totally independent and don't share memory.	A thread may share some memory with its peer threads.
Communication	Communication between processes requires more time than between threads.	Communication between threads requires less time than between processes .
Blocked	If a process gets blocked, remaining processes can continue execution.	If a user level thread gets blocked, all of its peer threads also get blocked.
Resource Consumption	Processes require more resources than threads.	Threads generally need less resources than processes.
Dependency	Individual processes are independent of each other.	Threads are parts of a process and so are dependent.
Data and Code sharing	Processes have independent data and code segments.	A thread shares the data segment, code segment, files etc. with its peer threads.
Treatment by OS	All the different processes are treated separately by the operating system.	All user level peer threads are treated as a single task by the operating system.
Time for creation	Processes require more time for creation.	Threads require less time for creation.

5. What is the Critical Section Problem and the three conditions for its solution.

Critical Section Problem :- Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any I/O device.

The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

The three condition for its solution :

I. **Mutual exclusion :-** If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

II. **Progress :-** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical

section next, and this selection cannot be postponed indefinitely.

III. **Bounded waiting :-** There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after has made a request to enter its critical section and before that request is granted.
We assume that each process is executing at a nonzero speed. However, we can make no assumption concerning the relative speed of the n processes.

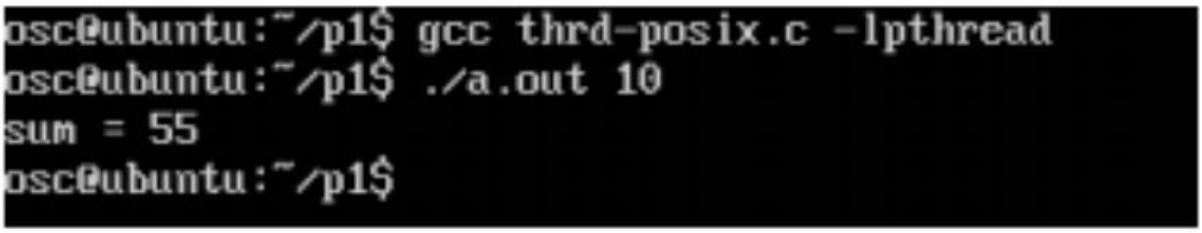
6. **Write a code for pthread sum of 2 non negative integer**.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
int sum;
void *runner(void *param);
int main(int argc, char *argv[])
{
	Pthread_t tid;
	pthread _attr_t attr;
	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;
	}

	Pthread_attr init(&attr);
	Pthread_create(&tid,&attr,runner,argv[1]);
	Pthread_join(tid,NULL);
	printf("sum = %d\n",sum);
}

void *runner(void *param)
{
	int i, upper = atoi(param);
	sum = 0;
```

```
    for (i = 1; i <= upper; i++)
    sum += i;
    pthread exit(0);
}
```



7. Write a code for POSIX scheduling.

```
#include <pthread.h>
#include <stdio.h>
#include<stdlib.h>
#define NUMTHREADS 5
int main(int argc, char *argv[])
{
	int i, policy;
	pthread_t tid[NUM THREADS];
	pthread_attr_t attr;
	pthread_attr_init(&attr);
	if (pthread_attr_getschedpolicy(&attr, &policy) != 0)
	fprintf(stderr, "Unable to get policy.\n");
	else
	{
		if (policy == SCHED_OTHER)
			printf("SCHED OTHER\n");
		else if (policy == SCHED_RR)
			printf("SCHED RR\n");
		else if (policy == SCHED_FIFO)
			printf("SCHED FIFO\n");
	}
	if (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) !=
0)
		fprintf(stderr, "Unable to set policy.\n");

	for (i = 0; i < NUMTHREADS; i++)
		pthread_create(&tid[i],&attr,runner,NULL);
	for (i = 0; i < NUMTHREADS; i++)
		pthread_join(tid[i], NULL);
}
void *runner(void *param)
```

```
{
        /* do some work ... */
        pthread exit(0);
}
```

8. Write the output of fork function.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int value = 5;
int main()
{
        pid t pid;
        pid = fork();
        if (pid == 0)
         {
                /* child process */
                value += 15;
                return 0;
        }

        else if (pid > 0)
         {
                /* parent process */
                wait(NULL);
                printf("PARENT: value = %d",value); /* LINE A
        */
                return 0;
        }
}
```

Output:- 5. As the child process is created using fork() system call, it creates separate memory with the same program as of the parent. So the increment of 15 is done in child's memory only. And line 5 is in Parent(main) memory so it will print 5.

9. **Write a program for pthread creation.**

```
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS 4
void hello (void *arg)
{
	printf("Hello Thread\n");
	return 0;
}

int main (void)
{
	int i;
	pthread_t tid[NUM_THREADS);
	for (i=0;i<NUM_THREADS; i++)
		pthread create(&tid [i], NULL, hello, NUL);
	for (i=0;i<NUM_THREADS; i++)
		pthread_join(tid[i], NULL);
	return 0;
}
```