# OS Theory Assignment 1

Suraj Telugu - CS20BTECH11050

## 1 Question 1

Problem 1.20 on page EX-2 of the book (page 84 of the pdf): Consider an SMP system similar to the one shown in Figure 1.8. Illustrate with an example how data residing in memory could in fact have a different value in each of the local caches.

**Answer:**
Symmetric Multi Processing (SMP) : Multi Processing in which every processor can perform all kinds of tasks including OS processes and user processes is called symmetric (or tightly coupled) multiprocessing (SMP). In SMP processors share I/O bus, Operating system monitors all the processors

In a SMP, Let there be a data residing in memory which is used in two different processes process 0, process 1 which execute in processor 0 and processor 1 respectively then

1) The data first gets copied to local cache of each processor from main memory
2) Later Operating system starts process scheduling without loss of generality assume process 0 processing starts in processor 0 first
3) During the process the data in local cache gets copied into registers(cache to local cache) and undergoes procedure,during the procedure the value of data can change and new value of data is stored in register
4) This value gets copied into local cache of processor 0 now we find that processor 0 and processor 1 have different values of data in their local caches
5) Later even process 1 might start and change the value of data and the new values of data in local caches may differ even more and even differ from original value of data

**Example:** Let data in memory be 20 and it gets copied into local caches of two processors (processor 0 and processor 1)

1) process 0 : Finds reminder of data when divided by 6 and returns the value of reminder into the data
2) process 1 ; Uses the value of data to create an array of size corresponding to the value of data returns data without changing

Note : in process 1 value of data does not change before and after processing

1) Before processing : Both local caches have 20 which was taken from main memory
2) After process 0 finishes in processor 0 ; local cache of processor 0 has value 2 and local cache of processor 1 has 20
3) after both processes finish: Local cache of processor 0 has value 2 and local cache of processor 1 still has value 20 since it did not change while undergoing the process

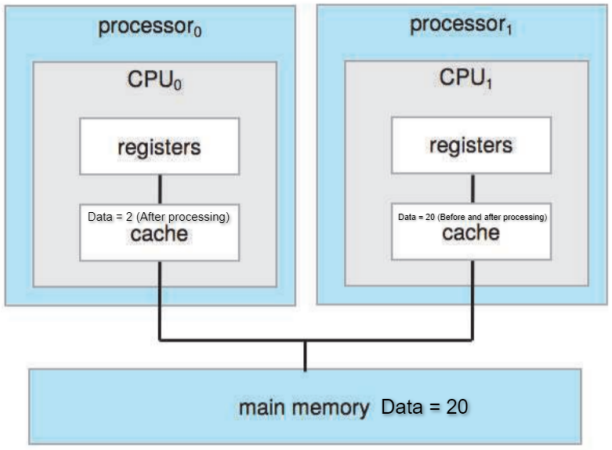


Fig. 1: Fig 1.8: Example (Symmetric Multiprocessing Architecture)

## 2 Question 2

Find out about the system calls used by printf and scanf functions. You can refer to any source from the Internet. But please cite your source in the answer.

**Answer:**
System call used by scanf : read()
System call used by printf : write()

read() : Read from file descriptor

1) Syntax : ssize_t read(int fd, void *buf,size_t count); , where ssize_t, size_t are signed, unsigned integer respectively, fd is file descriptor, *buf is local buffer address
2) read() system call reads maximum count number of bytes from fd and stores data in *buf and returns the number of bytes it read on error or signal interrupt -1 is returned
3) In scanf function it internally calls read() system call which uses fd = 0 which is standard input (stdin) once the input is taken from stdin it gets stored in local buffer whose address is the address of variable where data needs to be stored
4) Calls to read() makes a 'context switch' which takes it from user-level mode into kernel mode, it performs privileged operations, such as talking directly to hardware (keyboard). Using 'stdin' fd it gets input from keyboard

write() : write to a file descriptor

1) Syntax : ssize_t write(int fd, const void *buf, size_t count); , where ssize_t, size_t are signed, unsigned integer respectively, fd is file descriptor, *buf is local buffer address
2) write() system call writes up to count bytes into fd from the data in *buf and returns number of bytes written on success, -1 is returned if an error or signal interrupt is occurred
3) In printf function it internally calls write() system call which uses fd = 1 which is standard output (stdout), Input is taken from local buffer data is written into the stdout
4) Calls to write() makes a 'context switch' out of your user-level application into kernel mode, it performs privileged operations, such as talking directly to hardware (monitor). Using 'stdout' fd it displays output on monitor

Note : For both printf() and scanf() in their corresponding system calls the count is set to maximum its maximum limit

**References :**


1) Man pages of read, write system calls in Linux
2) https://profile.iiita.ac.in/bibhas.ghoshal/OS_2019/04-syscalls.pdf
3) https://stackoverflow.com/questions/1253132/how-does-scanf-work-inside-the-os/1253144


## 3 Question 3

Please study the PCB (Process Control Block) structure of a Linux system from any reliable Internet source. Describe any 10 fields in the PCB of a process in the latest Linux Operating System. Please cite your source as well like in question 2.

**Answer:**
Process Control Block: A process in an operating system is represented by a data structure known as a process control block (PCB/ Task Controlling Block / Task Struct). The PCB contains important information about the specific process including the current state of the process

The PCB of a scheduling process (sched.h) of Linux system is studied and the following fields of PCB are explored

1) **Volatile long state** : This field describes the current state of a process i.e whether it is ready, running or waiting
2) **long counter** : This field stores the value of program counter of process
3) **long priority** : This field handles the priority numbers of different processes
4) **int pid** : This field stores the process id for each process which is unique for every process
5) **struct task_struct p_opptr** : This field stores the pointer of original parent (oppr = Orginal Parent Pointer)
6) **struct task_struct p_cptr** : This field stores the pointer of most recent child process of original process (cptr = Child Pointer)
7) **struct task_struct *next_run** : This field stores struct pointer of next process in the queue in ready state

8) **struct task_struct *next_task** : This field stores struct pointer of next thread ready to start in the process
9) **long stime** : This field gives the system time of the process measured by system clock from start time (long start_time)
10) **int lock_depth** : It indicates how many times has been acquired the kernel lock. It is used to protect some of the kernel data structures.

All the above fields in PCB help the operating system to get data on the process and perform process scheduling, dealing with time and memory etc

**References :**

1) https://www.cs.fsu.edu/~zwang/files/cop4610/Fall2016/chapter3.pdf
2) http://lxr.linux.no/linux+v3.2.35/include/linux/sched.h#L1221

LaTeX generated document