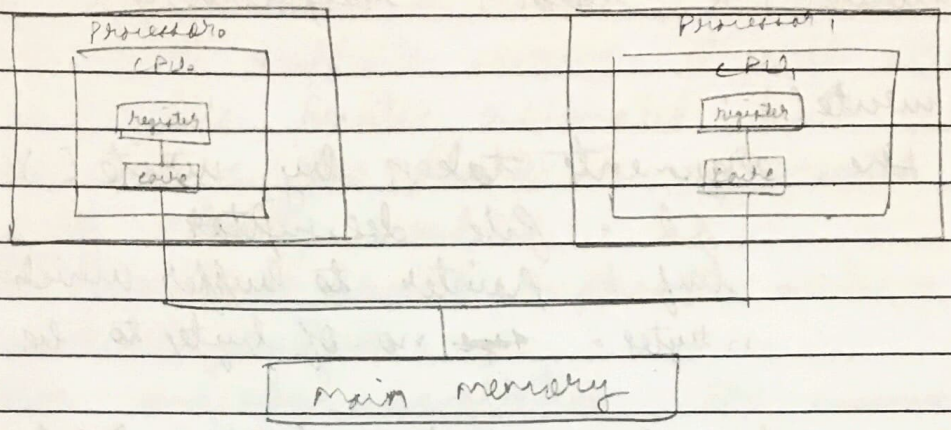
CS20BTECH11063 Tanmay Garg

OS-1 Assignment 1



- Let us say Processor 0 reads some data 'A' with some value 'a' from main memory into its own local cache
- Let us say Processor 1 also does the same
- Let Processor 1 update 'A' to 'b'
- Now, 'A' resides in Processor 1's cache, the update only occurs there
- So the Processor 0, the value of 'A' is still 'a'.
- This happens because for a particular process when Processor 1 loads the variable from memory & puts in its own cache & for some another process in Processor 0 loads the same variable & puts it in its own cache.
- When Processor 1 makes an update, that update doesn't take place globally as Processor 0 still has an older value for that variable

2. • The system calls used by printf & scanf is write() & read() respectively.

→ write():

• The arguments taken by write():
  • fd = file descriptor
  • buf = pointer to buffer which has the data
  • n bytes = no. of bytes to be written

• It writes n bytes of data to the object referenced by fd from buffer.
• The write() starts at a position given by the pointer associated with fd for objects that can perform seek functions.
• After return, pointer is incremented by no. of bytes that were written
• Objects that are not capable of seeking write from the current position
• If the function fails to write, then it returns -1.

→ read():

• The arguments taken by read():
  • fd = file descriptor
  • buf = pointer to buffer which has the data
  • n bytes = no. of bytes to be read.

• It reads n bytes of data from object referenced by fd into the buffer.

- Before any action is taken, if nbytes is 0, then may detect & return errors, or else it will return 0.
- For files that support seeking, it will start at a position in the file pointer associated with fd.
- Upon return, the pointer is incremented by the no. of bytes actually read.
- For files that don't support seeking start from current position
- Upon successful completion, the number of bytes actually read are returned.

- The system calls cause a 'context switch' out of user-level application into kernel mode
- The string conversion occurs in user mode and all other optimizations are done and then it goes into kernel mode
- Moving to kernel mode requires a lot switching of changing pointer to the current register set which is computationally expensive
- Hence, switching back & forth to kernel mode is made as low as possible for a particular type of function

Resources
- pubs.opengroup.org/onlinepubs/9699919799
- unix.com/man-page/freebsd/2/read
- Operating system concepts 10th e, ch 2
- man pages

3. • PCB is called a Process control block, each process is represented in the operating system using this.

• The fields of a PCB are :-

1. Process state : It signifies the current state of the process which can be in waiting, running, ready, new, halted and so on

2. Program counter : It indicates the address of the next instruction to be executed in the process

3. CPU registers : whenever there is a context switch between the processes, temporary information is stored in the registers. so when the process resumes, it continues exactly from where it swapped & continue correctly afterward.

4. CPU scheduling information : It includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
It may also be the case that the parameters for changing the priority of the process can be the age of that process etc.

5. Memory Management information : It may include information such as value of the base & limit registers & the page tables or the segment tables, depending on the memory system used by the OS.

6. Accounting information : It includes the amount of CPU & real time used, time limits

account numbers, job or process numbers etc. It gives the description of the resources used by that process.

7. I/O status information : It includes the list of I/O devices allocated to the process, during its execution.

8. List of open files : It contains the information of all the files that is required by the program during its execution.

9. Process ID : Whenever a new process is created, the OS allots a number to that process. This is a unique identification of that process & helps it distinguish from other processes.

10. PCB Pointer : It is the address of the next PCB, who process state is ready. The OS maintains the hierarchy of all the processes so that a parent process could locate all the child processes.

11. Event information : It contains the information of the event for which the certain process is in block state. If the event occured matches with this field the process changes its state from blocked to ready.

Resources

- binarytetons.com / process-control-block-pcb.html
- tutorialspoint.com / what-is-process-control-block-pcb
- Operating systems concepts, 10th e, Ch 3