Processes

1 . Kernel maintains data structure for process
- list of all procs
- mem. Management details of each file opened by each
- scheduling information about the proc
- status of proc

- list of proc “waiting” for diff events to occur

Process control Block

- record representing proc in OS’s data structure
- OS maintains list of PCBs
- “struct task_struct” in linux
  “struct proc” in xv6

Fields in PCB :

1. pid : id for each proc.
2. proc. State
3. prog. Counter
4. registers
5. Mem. Limits of the proc
6. Accounting info
7. I/o status
8. Scheduling info.
9. Array of file descriptor(fd) : list of open files

list of open files :
fd : return value of open (index in array of pointer which is maintained in pcb)

first null pointer in fd array, points to the proc.
fd[0] : standard input read(0,...) ~= scanf()
fd[0] : standard output write(1,...) ~= fwrite()
fd[0] : standard error

Process : code + data + stack + heap
Everythong else in kernel

struct ptable contains array of proc

Diff. Types of queues/list can be maintained by OS for the proc.
- queue of proc which need to be scheduled
- queue of proc which have requested ip/op to device and hence need to be put on hold/wait
- list of proc currently running on multiple CPUs

in Linux :
PCB : task_struct
consists of list_head (doubly linked list)which points to other list heads in other procs

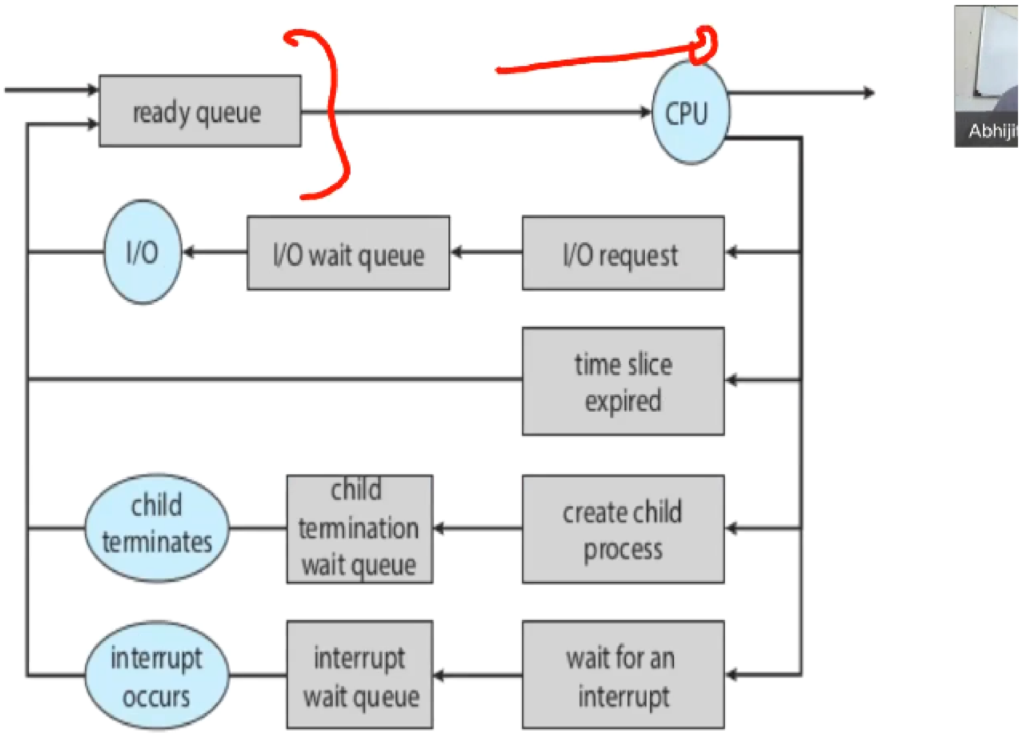


Figure 3.5 Queueing-diagram representation of process scheduling.

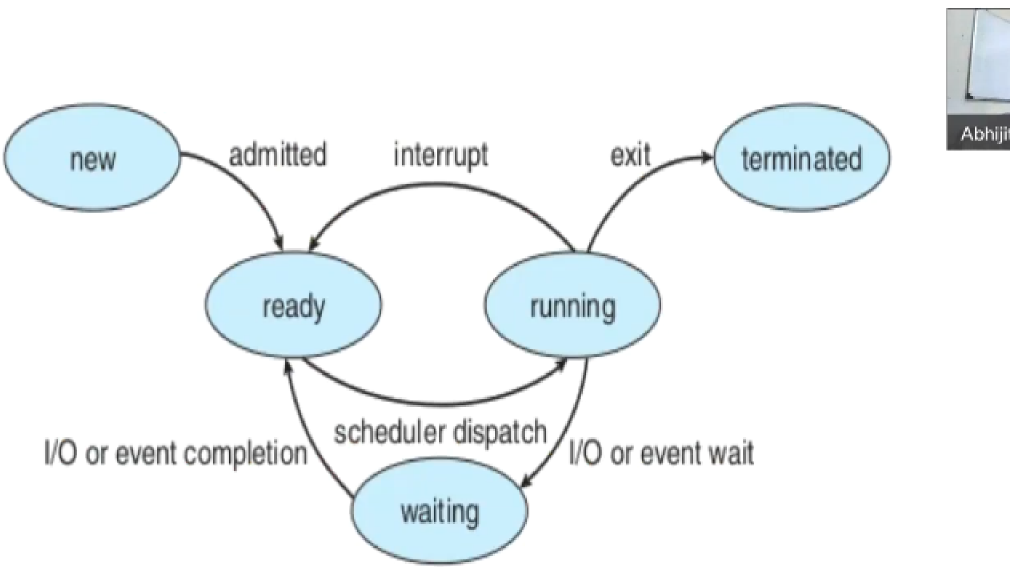
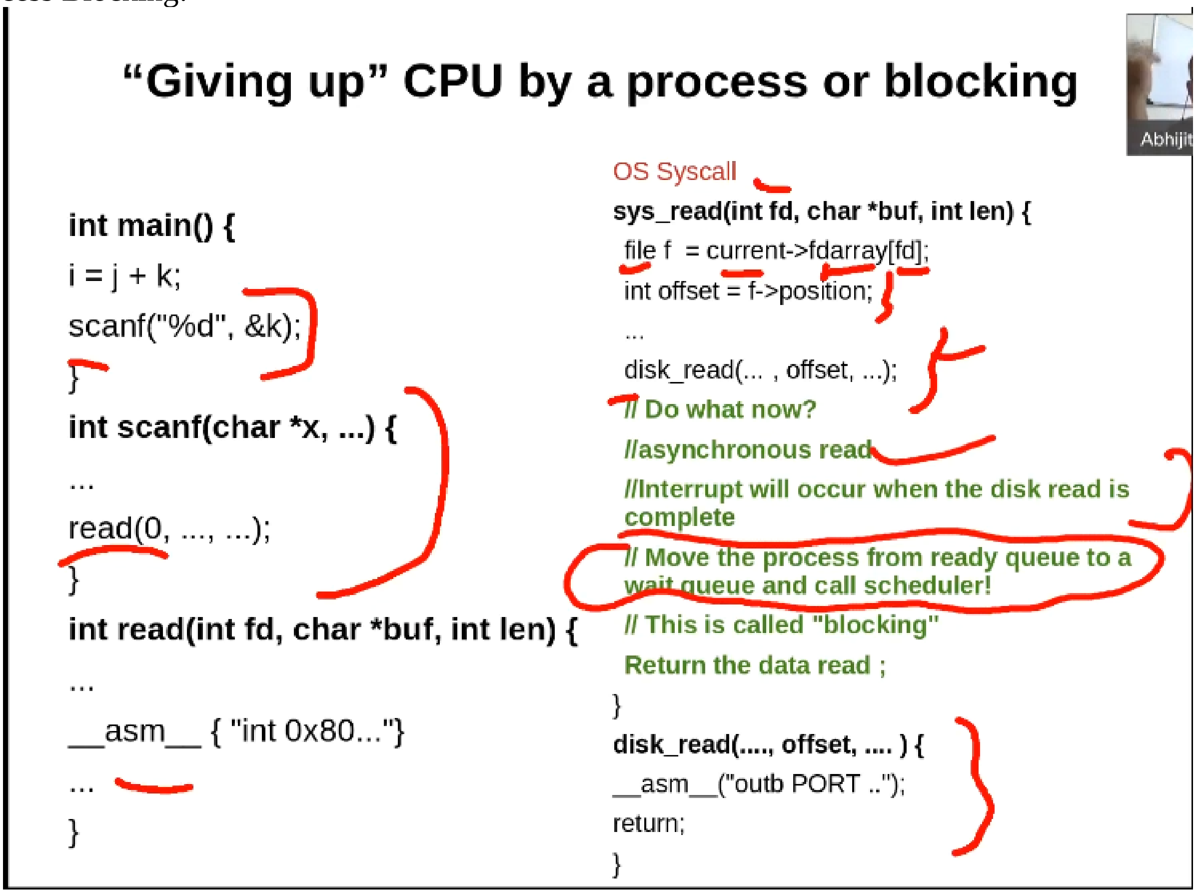


Figure 3.2 Diagram of process state.

Conceptual diagram

A process can only terminated by running **exit** .

Process Blocking:



##########################

Context swithing

Context:
- execution context of proc
- CPU reg, proc state, memory management info, all config of CPU that are specific to execution of proc/kernel

Context switch :
- change the context from one proc/OS to OS/another proc
- need to save the old context and load new context
- save in the **PCB of proc**

- Overhead
- no useful work is done during context switch
- time varies on h/w
- special instr, availeble to save set of reg in one go

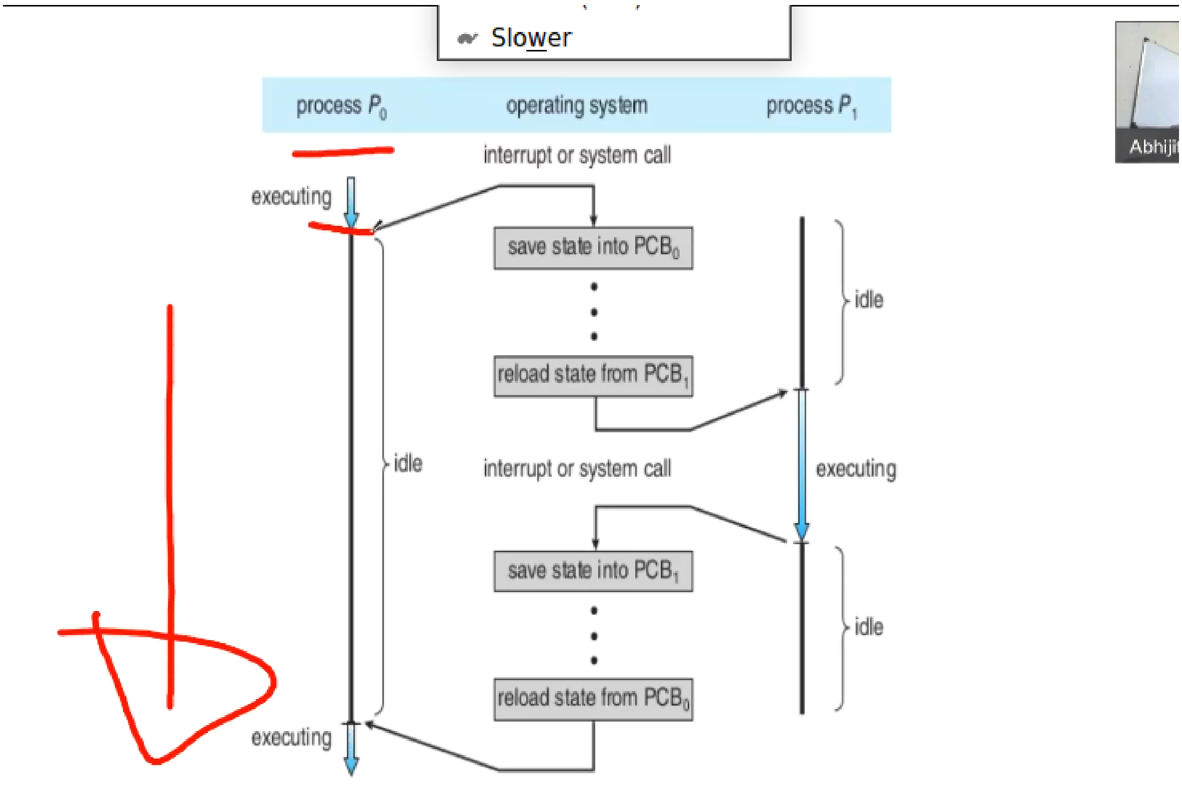


Figure 3.6 Diagram showing context switch from process to process.

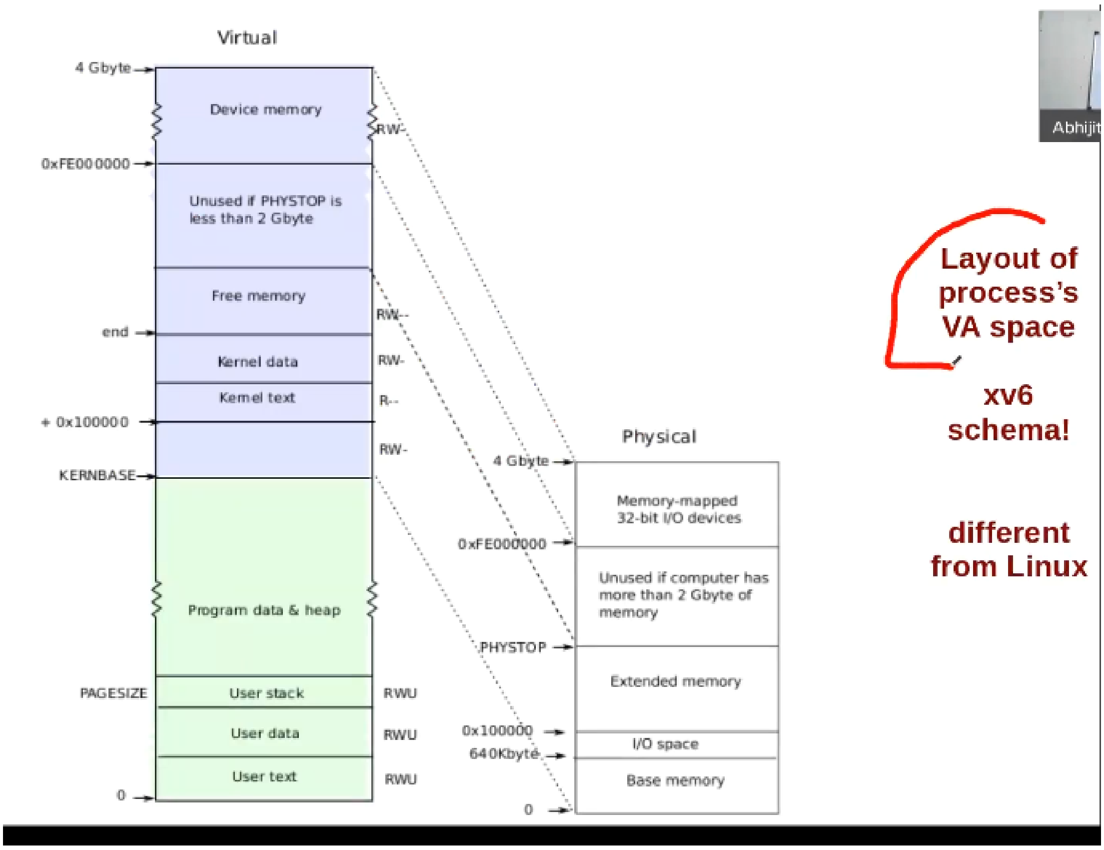
during triple section, kernel code is used.
Hence during each switch, switching to kernel and proc are also done (hence 2 switches)

Pecularity of context switch

- when proc is running, the func call are in LIFO order , possible due to calling convention
- interrupt are at any time, **Context switch happen in middle of execution of any funct**
- After contxt swtch, **one proc takes place of another**
- **in calling convention, function are in the same process**
- swtch is not happening using calling convention, No call is done.
- **code for contxt swtch is in assembly.** Because if it is C, then calling convention takes place which we want to be violated here

##########################

Processes in xv6:



Suppose code is at address 0
followed by data 4K
(something else is here) **Guard Page**
followed by stack
KERNBASE : 0x80000000 (2GB)

Assumed that kernel is loaded in virtual address by KERNBASE onwards
(2GB + 1MB onwards kernel code + data)
Then free memory

When we do exec, we have to mapping

1. Address 0: code
2. Then globals
3. then stack
4. then heap
5. Every proc. Addr. Space mapy s kernel txt, data also
6. so sys. Call run with these mapping
7. Kernel code can directly access user data.

Kernel : loaded at 0x10000 Physical addr.
BIOS and Devices : Phy. Addr. 0 to 0x100000

Process page table maps to
0x80000000 – 0x8010000 : To 0x00000 – 0x10000

Kernel is not loaded at Phy. Addr. 0X8000000 As some sys. MAY NOT have much memory

Process has 2 stacks
1. User stack : Used when user code is running
2. Kernel stack : when kernel is runnig behalf of proc.

*3. Kernel stack : uses kernel stack when scheduler is running. **Not per proc.**

in Struct proc:
size of proc < 2gb //available in elf file
pgdir : Xv6 : uses 2 level paging
    when scheduling pointer goes to CR3
    points to base of frame hosting top level page directory which will further point to the base of page frame (0 – 4GB)
context : points to location in kernel where context is stored

Guard page:
    mapping in page table
    no frames are allocated
    marked invalid
    - if stack grows in size (due to function calls), OS gives exception