Context Switching and Process Scheduling

Context Switch

- PC, processor status word, registers, pushed onto a small kernel stack allocated for the process
- Jump to routine in kernel indicated by trap table/interrupt vector
- Kernel services the trap/interrupt
- Kernel selects the next process to run
  - It may be a different one
- Kernel pops the information from the kernel stack and restores them to the registers, processor status word, and PC
- PC popped last, as when it is restored, process restarts

Context Switch Example

- Example is from XINU on an LSI-11 system
  - See the handout "Context Switch Routine for XINU System on LSI-11"
  - LSI-11 has same instruction set as PDP-11
- Modern systems may have more complex entries, but the idea is the same

Limited Direct Execution

operating system at boot (kernel mode)	hardware	program/process (user mode)
initialize interrupt/trap vectors		
	remember address of syscall handler	
create entry for process in process table		
allocate memory for program		
load program into memory		
setup stack (including arguments, environment, etc.)		
fill kernel stack with registers, PC		
return-from-trap		
	restore registers from kernel stack	

Limited Direct Execution

operating system at boot (kernel mode)	hardware	program/process (user mode)
	change to user mode	
	jump to **main()**	
		run **main()**
		system call causes trap into OS
	save registers to kernel stack	
	change to kernel mode	
	jump to trap handler	
handle trap, ie system call		
return-from-trap		
	restore registers from kernel stack	

Limited Direct Execution

operating system at boot (kernel mode)	hardware	program/process (user mode)
	change to user mode	
	jump to PC after trap	
		return from **main()**
		causes trap
free memory of process		
delete process table entry		

Process Scheduling

- Depends on what you want from the system
- Metrics are:
  - Throughput: get the most work done in a given time
  - Turnaround: complete processes as soon as possible after submission
  - Response: minimize the time between submission and first response; does not include time to output the response
  - Resource use: keep each type of resource assigned to some process as much as possible, but avoid waiting too long for certain resources.
  - Waiting time: minimize the amount of time the process sits in the ready queue
  - Consistency: treat processes with given characteristics in a predictable manner that doesn't vary greatly over time.

Scheduling

- Many attributes affect this:
  - priority
  - anticipated resource need (including running time)
  - running time, resources used so far
  - interactive/non-interactive
  - frequency of I/O requests
  - time spent waiting for service

Comparing Scheduling Algorithms

Job	Arrival time	Service time
A	0	10
B	1	29
C	2	3
D	3	7
E	4	12

Measures:

- Turnaround time: time the process is present in the system
  T = finish time – arrival time
- Waiting time: time the process is present and not running
  W = T - service time
- Response ratio: the factor by which the processing rate is reduced, from the user's point of view: R = T / service time

Types of Scheduling Algorithms

- Decision mode:
  - Non-preemptive: a process runs until it blocks or completes; at no time during its run will the operating system replace it with another job
  - Preemptive: the operating system can interrupt the currently running process to start another one
- Priority function: a mathematical function that assigns a priority to the process
  - Process with the highest (numerical) priority goes next
  - Function usually involves the service time so far a, the real time spent in the system so far r, and the total required service time t
- Arbitration rule: if two processes have the same priority, this rule states how one of them is selected to run.

Process (Job) Scheduling

- First In, First Out (FIFO)
- Shortest Job (Process) Next/First (SJN, SJF, SPN, SPF)
- Highest Response Ratio Next (HRRN)
- Round Robin (RR)
- Multi-Level Feedback Queue (MLFQ)

First Come First Serve

- Decision mode: non-preemptive
- Priority function: $p(a, r, t) = r$
- Arbitration rule: random

Process	Service time	Arrival time	Start	Finish	T	W	R
A	10	0	0	10	10	0	1.0
B	29	1	10	39	38	9	1.3
C	3	2	39	42	40	37	13.3
D	7	3	42	49	46	39	6.6
E	12	4	49	61	57	45	4.8
mean					38.2	26	5.4

As a Graph

Problem

- Sensitive to order jobs arrive
- Example:

Process	Service time	Arrival time	Start	Finish	T	W	R
A’	1000	0	0	1000	1000	0	1.0
B’	1	1	1000	1001	1000	999	1000.0

But:

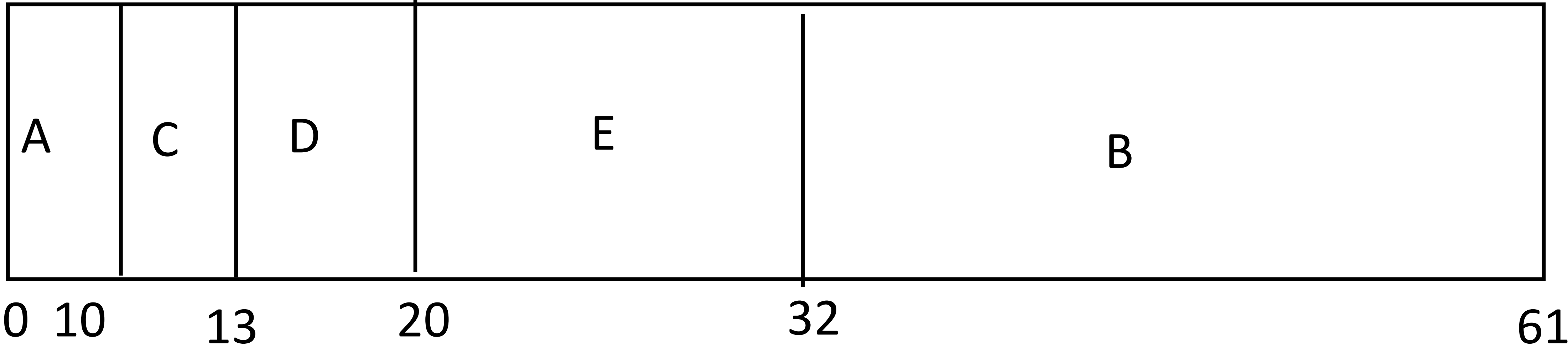
Process	Service time	Arrival time	Start	Finish	T	W	R
B’	1	0	0	1	1	0	1.0
A’	1000	1	1	1001	1000	1	1.0

Shortest Process Next

- Decision mode: non-preemptive
- Priority function: $p(a, r, t) = -t$
- Arbitration rule: chronological or random

Process	Service time	Arrival time	Start	Finish	T	W	R
A	10	0	0	10	10	0	1.0
B	29	1	32	61	60	31	2.1
C	3	2	10	13	11	8	3.7
D	7	3	13	20	17	39	2.4
E	12	4	20	32	28	10	2.3
mean					25.2	17.6	2.3

As a Graph



Shortest Process Next

- SPN gives the smallest average turnaround time out of all non-preemptive priority functions
- SPN better than FCFS for short jobs, but long jobs may have toi wait for some time for service

Problem

SPN needs to know service times into the future so it can run the process with the shortest next CPU burst. To choose the next process to run, it can use a number of different ways:

- Most accurate is to run all ready processes, time the CPU bursts, and then schedule them (*impractical*)
- Characterize each process as CPU-bound or I/O-bound, and specify for each an "average service time needed" based upon timing processes over a period of time and averaging. These characteristics might change over a period of time; that is, a process might be CPU-bound for a time, then I/O-bound, then CPU-bound, etc.

Problem

- Compute the expected time of the next CPU-burst as an exponential average of previous CPU-bursts of the process. Let t_n be the length of the *n*-th CPU burst, and t_{init} the initial estimate.

$$t_{n+1} = at_n + (1-a)t_{init}$$

where a is a parameter indicating how much to count past history (usually chosen around 0.5)

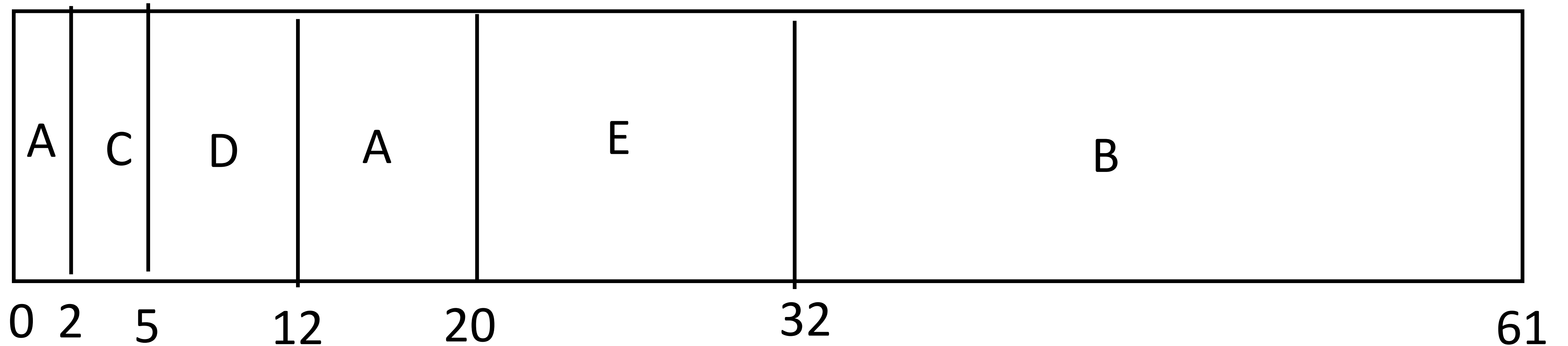
  - a = 1: the estimate is simply the length of the last CPU burst
  - a = 0: the estimate is the initial estimate

Shortest Remaining Time Next

- Decision mode: preemptive
- Priority function: $p(a, r, t) = a - t$
- Arbitration rule: chronological or random

Process	Service time	Arrival time	Start	Finish	T	W	R
A	10	0	0, 12	2, 20	20	10	2.0
B	29	1	32	61	60	31	2.1
C	3	2	2	5	3	0	1.0
D	7	3	5	12	9	2	1.3
E	12	4	20	32	28	16	2.3
mean					24	11.8	1.74

As a Graph



Highest Response Ratio Next

- Decision mode: non-preemptive
- Priority function: $p(a, r, t) = a/c$
- Arbitration rule: FIFO or random

Process	Service time	Arrival time	Start	Finish	T	W	R
A	10	0	0	10	10	0	1.0
B	29	1	32	61	60	31	2.1
C	3	2	10	13	11	8	3.7
D	7	3	13	20	17	10	2.4
E	12	4	20	32	28	16	2.3
mean					25.2	13	2.3

Why?

- To decide which process to run, compute:
  (estimated service time + waiting time) / estimated service time
- Idea: get mean response ration low, so if a process has a high response ratio, it should be run at once to reduce mean

Time	A	B	C	D	E	which runs
0						A
10		(29+9)/29=1.3	**(3+8)/3=3.7**	(7+7)/7=2.0	(12+6)/12=1.5	C
13		(29+12)/29=1.4		**(7+10)/7=2.4**	(12+9)/12=1.8	D
20		(29+19)/29=1.7			**(12+16)/12=2.3**	E
E		**(29+31)/29=2.1**				B

Round Robin

- Designed especially for time sharing
  - Uses *quantum*, typically between 1/60 sec and 1 sec
- Processes kept in a queue
- As each process is preempted, it moves to the rear of the queue
- All new arrivals come in at the rear of the queue

Example

- Using our previous jobs with a quantum of 5:

time	0	5	10	13	18	23	28	33	35	40	45	47	52	57	61
proc	A	B	C	D	E	A	B	D	E	B	E	B	B	B	
rem	5	24	0	2	7	0	19	0	2	14	0	9	4	0	

Variants

- Round Robin, but adjust quantum periodically.
  - *example*: after every process switch, the quantum becomes q/n, where n is the number of processes in the ready list
  - Few ready processes means that each gets a long quantum, minimizing process switches.
  - Lots of ready processes means that this algorithm gives more processes a shot at the CPU over a fixed period of time, at the price of more process switching
  - Processes needing a small amount of CPU time get a quantum fairly soon, and hence *may* finish sooner.
- Round Robin, but give the current process an extra quantum when a new process arrives
  - This reduces process switching in proportion to the number of processes arriving.