Interrupt handling: check for interrupts after every instr.-> save cur. context-> check interrupt vector table->disable interrupts->handle interrupts->enable interrupts-> select ready process and load into CPU

FCFS is simple and minimizes overhead.

If tasks are variable in size, then FCFS can have very poor ART

If tasks are equal in size, FCFS is optimal in terms of ART

SRTF becomes equivalent to FCFS

RR would do increase ART significantly

SRTF is optimal in terms of ART

The only way to implement is for kalman filter-like approximators for the individual CPU bursts

But NOT fair

We also do not have preemption – longer jobs can have very long waiting times, making them unresponsive

If tasks are equal in size, RR will have poor ART

RR works poorly on a mix of CPU and I/O bound tasks

SJF is hugely beneficial in this case

RR avoids starvation and is fair

To avoid the downsides of RR and SRTF, we want something in the middle

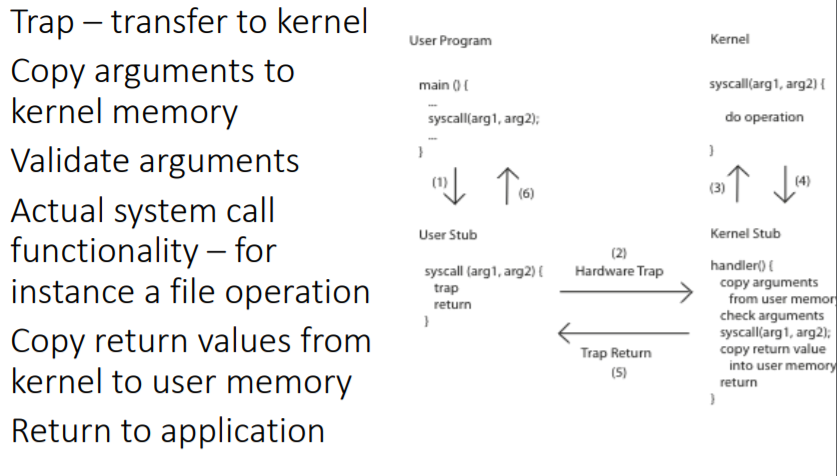
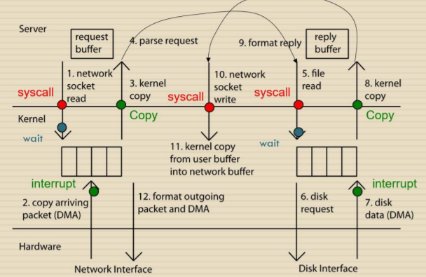
That would combine the good sides of both

MFQ scheduler is the most practical – answer to our prayer

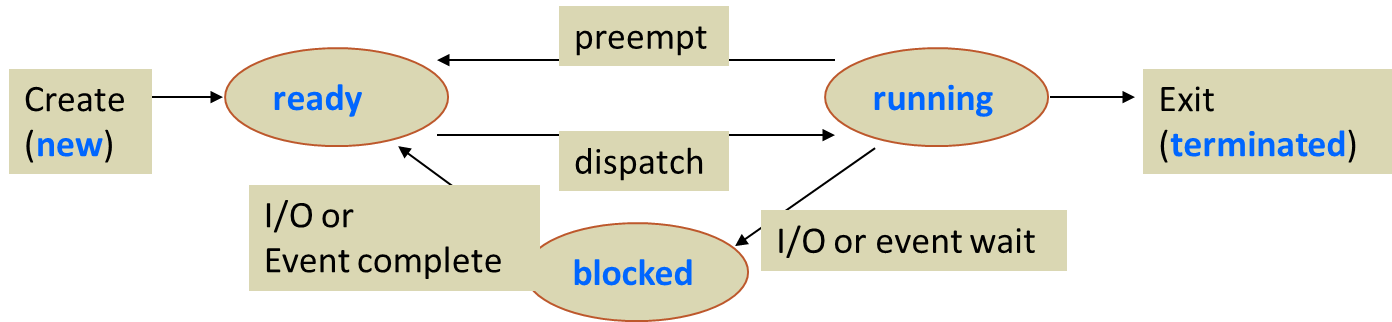
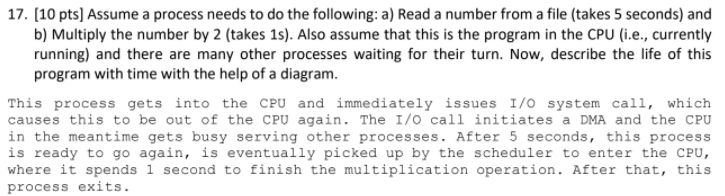
Approximates SJF while running just RR for each queue, no need to run any adaptive algorithm got the CPU bursts

Achieves a balance between responsiveness, low overhead, and fairness

Good for mixed workloads (user typing and long CPU computations running simultaneously)



Avg wait time=time to be executed-arival time Avg turnaround time (avg respond time)=completion time -arival time



Referee

Manage sharing of resources, Protection, Isolation

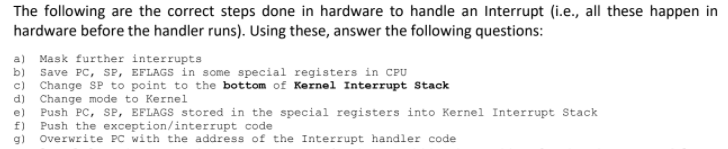
Resource allocation, isolation, communication

Illusionist

Provide clean, easy to use abstractions of physical resources

Infinite memory, dedicated machine

Masking limitations, virtualization

Glue

Common services

Storage, Window system, Networking

Sharing, Authorization

Look and feel