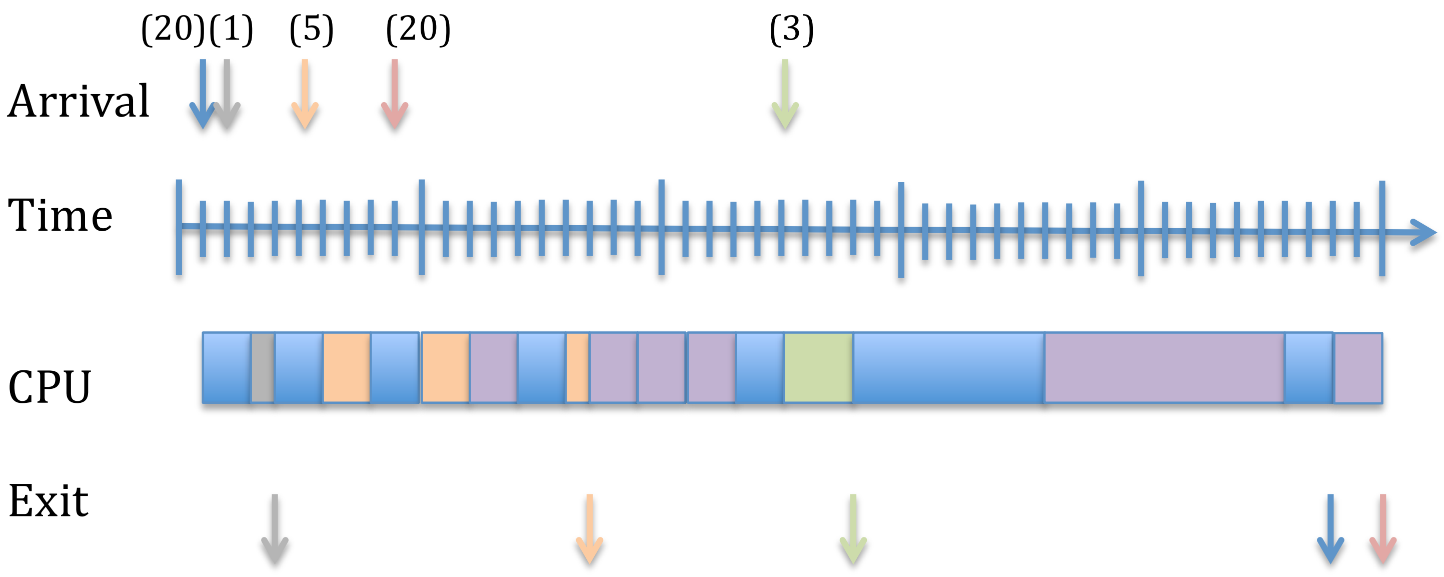
CS323-Exercise Solutions
Week 3
8/3/2019

Problem 1. Scheduling algorithms



Problem 2: Scheduling algorithms

A) P0 arrives first, so it will run first until completion

B) For example, a combination of Round Robin with time quantum 5 and Shortest Job First with preemption.

   a. Round Robin has higher priority and each process is first put at the end of this queue.
   b. After executing for 5s a process is preempted and put at the end of SJF scheduler queue.

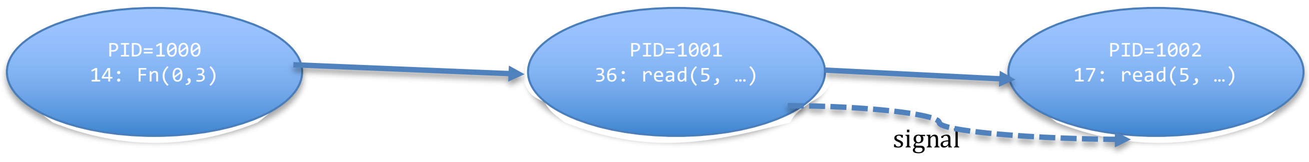
      This way:
      - P0 runs between 1 and 6
      - P1 runs between 6 and 7
      - P2 runs between 7 and 12
      - P3 runs between 12 and 17
      - P3 runs between 17 and 25 (shorter job in the SJF scheduler) P4 runs between 25 and 28
      - P3 runs between 28 and 30
      - P0 runs between 30 and 45

C) Shortest Job First, preemptive
   a. When P2 arrives at time 5, P1 is already finished and P1 has 16 seconds left
   b. Therefore, P2 starts immediately and ends at t=10s

Problem 3: IPC and signals

The initial process (PID=1000), creates a pipe (pd = [3,4]). It sets the signal handler for SIGUSR1 to be function h. It sets n = 4. It creates a child (PID = 1001) and sets p = 0. It blocks reading fd 3 in line 36.
Process 1001 creates another pipe (pd = [5,6]) and sets pd[1] to n. So pd = [5,4]. It creates a child (PID=1002). It sends a SIGUSR1 signal to the child. Then, it reads fd 5 and blocks in line 36.
Process 1002 executes the signal handler h. It writes 2 bytes in fd 4 and tries to read 2 bytes from fd 5, since it inherits the file descriptors from its parent. None has written to fd 5. So it blocks there.
Process 1000 wakes up because process 1002 wrote to fd 4. It reads a byte and calls Fn in line 37 with p = 0 and pd[0] = 3. Then it blocks inside Fn.

So, graphically we have the following:



Problem 4: Remote procedure calls

```
struct request{
	int Proc_id;
	char c;
	int len;
	char[] s;
}

struct response {
	int pos;
}
```

Client:

```
int_client_stub(char c, char*s)
{
	int len = strlen(s);
	int total_len = len + 1 + sizeof(struct request);

	Client_msg* msg = malloc(total_len);
	msg->proc_id = proc_no;
	msg->c= c;
	msg->len = len;
	strcpy(msg->s, s);

	reponse* res;

	Send(msg, total_len);
	res = Receive();

	return res->pos;
}
```

Server:

```
void_stub(proc_no, msg)
{
	struct_msg* = Receive();
	int return_val = call_server function (msg->proc_no, msg->c, msg->s);

	response resp* r = malloc(struct response);
	r->pos = return_val;
	Send (r, sizeof(struct response));
}
```