CSC112 Operating System- Assignment 2

Agnes Duru

```
…
#define NUM_THREADS 4
char * numbers[]={"One", "Two", "Three", "Four"};

void * worker(void * param){
    long x = rand()%4;
    printf("%s, %s\n", (char*)param, numbers[x]);
    return (void *)x;
}
int main(int argc, char * argv[]){
    pthread_t tid[NUM_THREADS];
    void * status;
    srand(time(0));                                    /* LINE A */

    if(fork()== 0){
        fork();
        for(int i=0; i<NUM_THREADS; i++){
            pthread_create(&tid[i], NULL, worker,(void *)numbers[i]);
        }
    }
    else{
        if (fork() != 0){
            pthread_create(&tid[0], NULL, worker, "Zero");
            pthread_join(tid[0], &status);
            printf("status: %ld\n", (long)status);
        }
    }
    printf("Done\n");
    pthread_exit(0);                                   /* LINE B */
}
```

Parent Process Creates Child A

Child A creates Child B

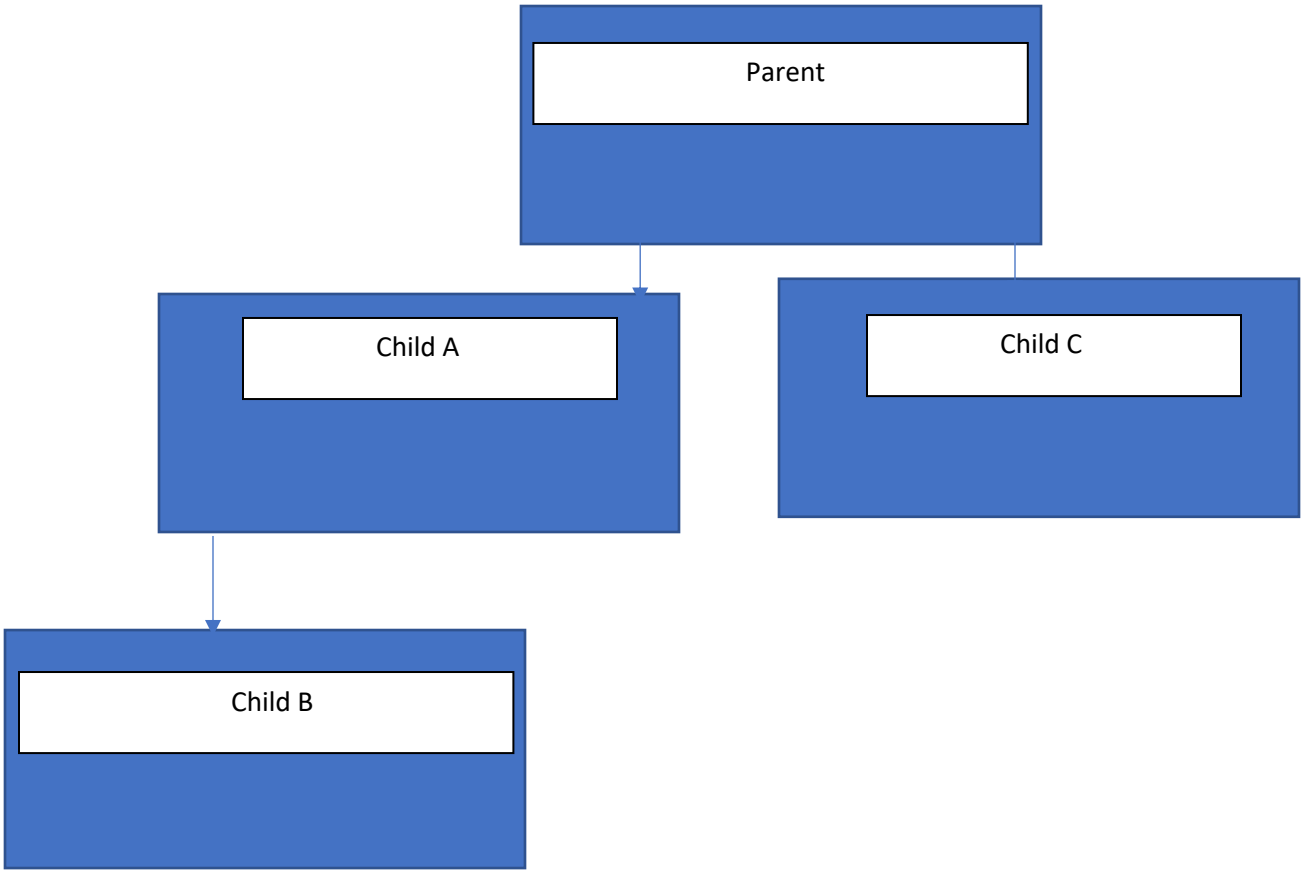
Child

Parent Creates Child C

Parent

All

i) How many unique processes are created by this snippet of code (including the original one)? Draw the tree of processes to show the relationship between the processes.



a. Answer: 4 processes are created by this snippet of code. I have included the relationship above.

ii) Indicate the threads in each one of the processes (from (i)) that are created because of the code snippet above?

- Answer: Parent uses the main thread and creates Thread 0. Child C creates thread 0. Child A uses its main thread to create Thread 0, Thread 1, Thread 2, and Thread 3. Child B uses its main thread to create Thread 0, Thread 1, Thread 2, and Thread 3.

iii) Identify the memory segment where each one of the variables tid, status, x and numbers will be “stored”. -- Note you have more than one threads, so you need to specify what variables will be stored to the unique segment(s) of each thread.

- Answer: Child A's threads 0,1,2,3 and Child B's threads 0,1,2,3 would have access to the variables tid, status, x and numbers from their individual stacks. Parents main thread would have access to tid, status, and numbers from its stack. Parent's thread 0 and Child C's thread 0 would have access to the variables tid, status, x and numbers. In the processes the variables tid, stack would be stored in the stack segment. Numbers would be stored in the Data segment.

iv) Briefly explain how the status variable will get its value(s).

- Answer: The status variable will get it values depending on whether or not the thread at index 0 was able to join successfully.

v) Describe the final possible output(s)? Since the rand() function is being used you can pick random integers (in the proper range) to indicate the output of .

- Answer: The finial possible outputs can be:
  Outputs:

- (1.a)-Child A can complete its threads and then child B completes all their threads. Child A states Done before B.

  Child A would print out:
  Thread 0 would print out: Zero, 0ne
  Thread 1 would print out: One, Three
  Thread 2 would print out: Zero, Two
  Thread 3 would print out: One, Four

  Child B would print out:
  Thread 0 would print out: Zero, Two
  Thread 1 would print out: One, One
  Thread 2 would print out: Zero, Four
  Thread 3 would print out: Zero, Three

  Child C would print out:
  Prints: Zero, Four
  Prints: status: 0
  Prints: Done

  Parent would print out:
  Prints: Zero, One
  Prints: status: 0

  Child A would print out:
  Done

Child B would print out:
Done

Parent:
Done

- (1.b)-Child A can complete its threads and then child B completes all their threads. Child B states Done before A

Child A would print out:
Thread 0 would print out: Zero, 0ne
Thread 1 would print out: One, Three
Thread 2 would print out: Zero, Two
Thread 3 would print out: Zero, Four

Child B would print out:
Thread 0 would print out: Zero, Two
Thread 1 would print out: One, One
Thread 2 would print out: Zero, Four
Thread 3 would print out: Zero, Three

Child C would print out:
Prints: Zero, Four
Prints: status: 0
Prints: Done

Parent would print out:
Prints: Zero, One
Prints: status: 0

Child B would print out:
Done

Child A would print out:
Done

Parent:
Done

- (2.a) Child A and Child B can alternate and Child A states Done before B.

Childs A and B alternating:

Child A: Thread 0 would print out: Zero, 0ne
Child B: Thread 0 would print out: Zero, Two
Child A: Thread 1 would print out: One, Three
Child B: Thread 1 would print out: One, One
Child A: Thread 2 would print out: Zero, Two
Child B: Thread 2 would print out: Zero, Four
Child A: Thread 3 would print out: Zero, Four
Child B: Thread 3 would print out: Zero Three

Child C would print out:
Prints: Zero, Four
Prints: status: 0
Prints: Done

Parent would print out:
Prints: Zero, One
Prints: status: 0

Child A would print out:
Done

Child B would print out:
Done

Parent:
Done

- (2b) Child A and Child B can alternate and Child B states Done before A
  Childs A and B alternating:

Childs A and B alternating:

Child A: Thread 0 would print out: Zero, 0ne
Child B: Thread 0 would print out: Zero, Two
Child A: Thread 1 would print out: One, Three
Child B: Thread 1 would print out: One, One
Child A: Thread 2 would print out: Zero, Two
Child B: Thread 2 would print out: Zero, Four
Child A: Thread 3 would print out: Zero, Four
Child B: Thread 3 would print out: Zero, Three

Child C would print out:
Prints: status: 0
Prints: Done

Parent would print out:
Prints: Zero, One
Prints: status: 0

Child B would print out:
Done

Child A would print out:
Done

Parent:
Done

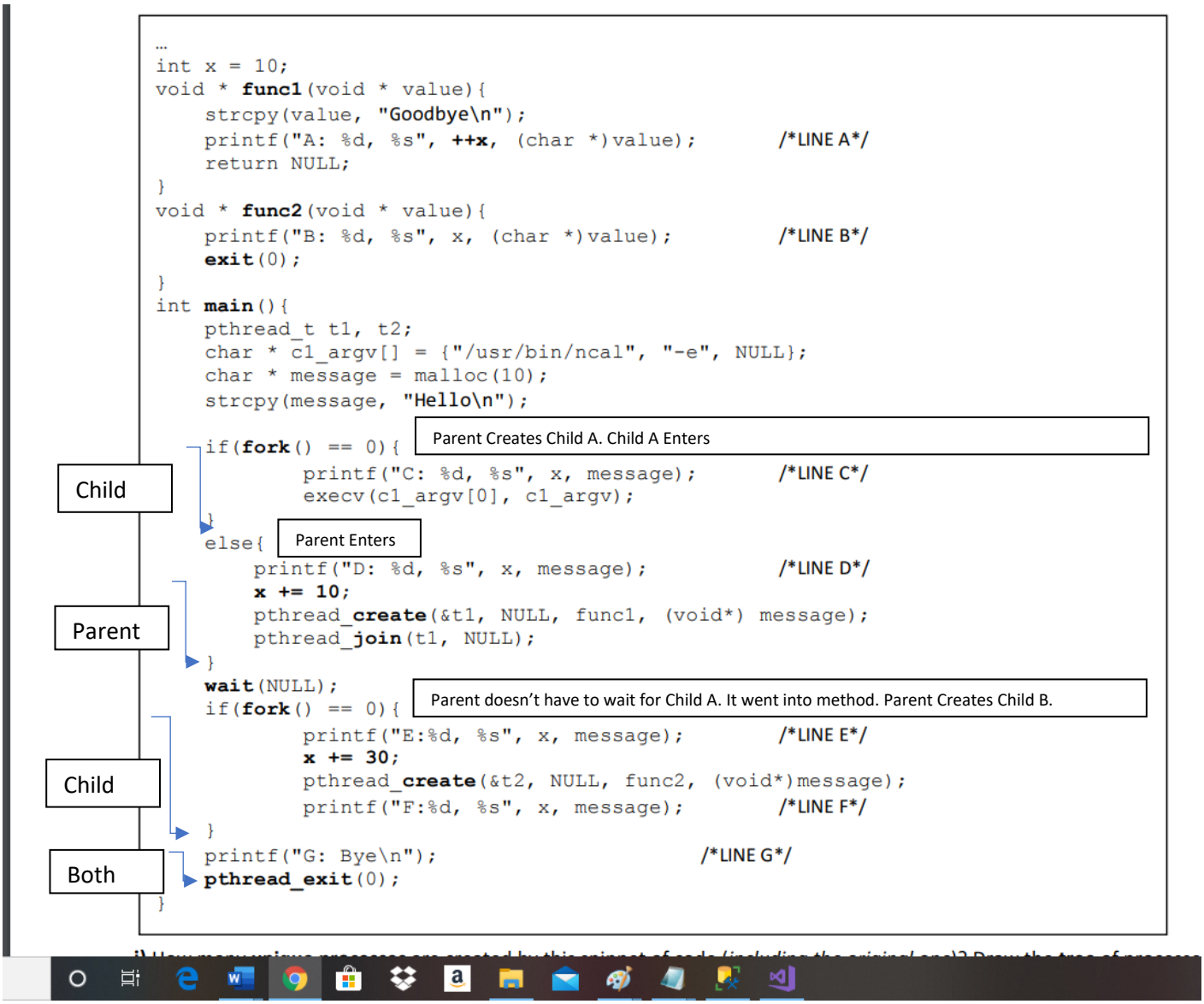
vi) What are the guarantees implied by the code above regarding the final output? Justify your answer. vii) Will your answers for (v) and (vii) change if Line B is replaced with "return 0;"? Justify your answer.

- Answer: Two guarantees are that 1. Because the pthread_join is being used the main thread will need to wait for specified thread thread 0 to finish. The 2nd guarantee is that the main thread would need to wait for all other threads to finish before it completely finishes because of the use of pthread_exit(0.) If pthread_exit(0) on LINE B was replaced with return 0 then the parent would be able to finish executing before the Child processes and would thus completely stop all threads that were running for that program.

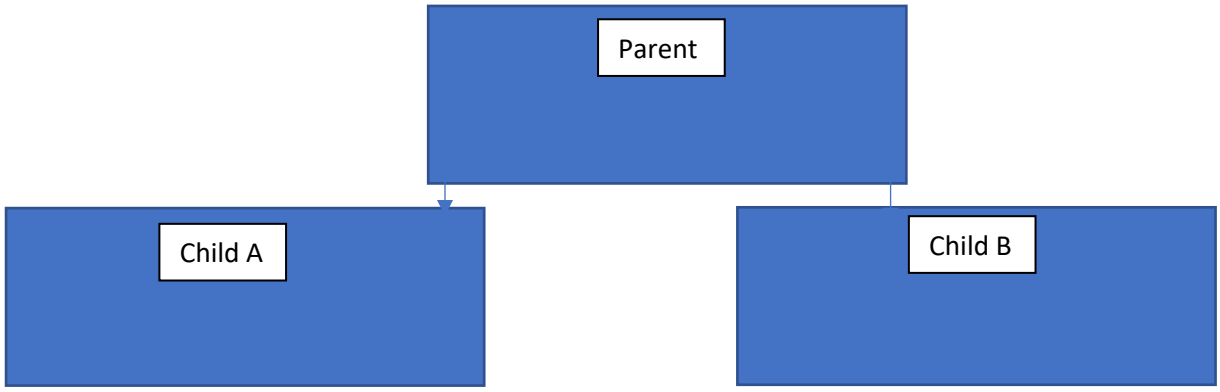
vii) What is the purpose of LINE A? Will something change if LINE A is commented out? Justify your answer.

- Answer: The purpose of LINE A is to have the pseudo-random number vary for every program call. If it is commented out then the program using the rand() function will create the same sequence of numbers on every program run. This is the case because the time() will return a time_t value which will vary everytime.

Question #2


…
int x = 10;
void * func1(void * value){
strcpy(value, "Goodbye\n");
printf("A: %d, %s", ++x, (char *)value); /*LINE A*/
return NULL;
}
void * func2(void * value){
printf("B: %d, %s", x, (char *)value); /*LINE B*/
exit(0);
}
int main(){
pthread_t t1, t2;
char * c1_argv[] = {"/usr/bin/ncal", "-e", NULL};
char * message = malloc(10);
strcpy(message, "Hello\n");
if(fork() == 0){
Parent Creates Child A. Child A Enters
Child
printf("C: %d, %s", x, message); /*LINE C*/
execv(c1_argv[0], c1_argv);
}
else{
Parent Enters
printf("D: %d, %s", x, message); /*LINE D*/
x += 10;
Parent
pthread_create(&t1, NULL, func1, (void*) message);
pthread_join(t1, NULL);
}
wait(NULL);
if(fork() == 0){
Parent doesn't have to wait for Child A. It went into method. Parent Creates Child B.
printf("E:%d, %s", x, message); /*LINE E*/
x += 30;
Child
pthread_create(&t2, NULL, func2, (void*)message);
printf("F:%d, %s", x, message); /*LINE F*/
}
printf("G: Bye\n"); /*LINE G*/
Both
pthread_exit(0);
}

i) How many unique processes are created by this snippet of code (including the original one)? Draw the tree of processes to show the relationship between the processes.



a. Answer: A total of three processes are created.

ii) Indicate the threads in each one of the processes that are created because of the code snippet above?

a. The parent used the main thread and created the Thread t1. Child A did not create any threads. Child B created the Thread t2.

iii) Identify the memory segment where each one of the variables t1, t2, message, and x will "store" its values. -- Note you have more than one threads, so you need to specify what variables will be stored to the unique segment(s) of each thread.

a. Answer: Parents main thread would have access to t1,t2, message and its thread t1 would have access to x and message from its stack. Child B's thread t2 would have access to message, and x from its stack. In the processes t1,t2, would be stored in the stack segment, message would be stored in the heap and x would be stored in the data segment.

iv) What are the possible outputs per process from this code? What are the guarantees implied by the code above regarding the final output (intra-process and inter-process)– e.g. what are the possible 1st line(s) of output and last line(s) of the final output? [**Inter-Process** communication means the data sharing between two different **processes**. where as, in **intra-process** communication the data shared with in components (methods) of a single **process**.]

a. Answer:

Output: Option A with the thread t2 being created before the print statement

Child A will print out → C: 10, Hello

Child A executes EXECV and gets list of argument/option values.
Parent will print→ D: 10, Hello
→A: 21 Goodbye
Child B will print→ E: 21, Goodbye
B: 51, Goodbye

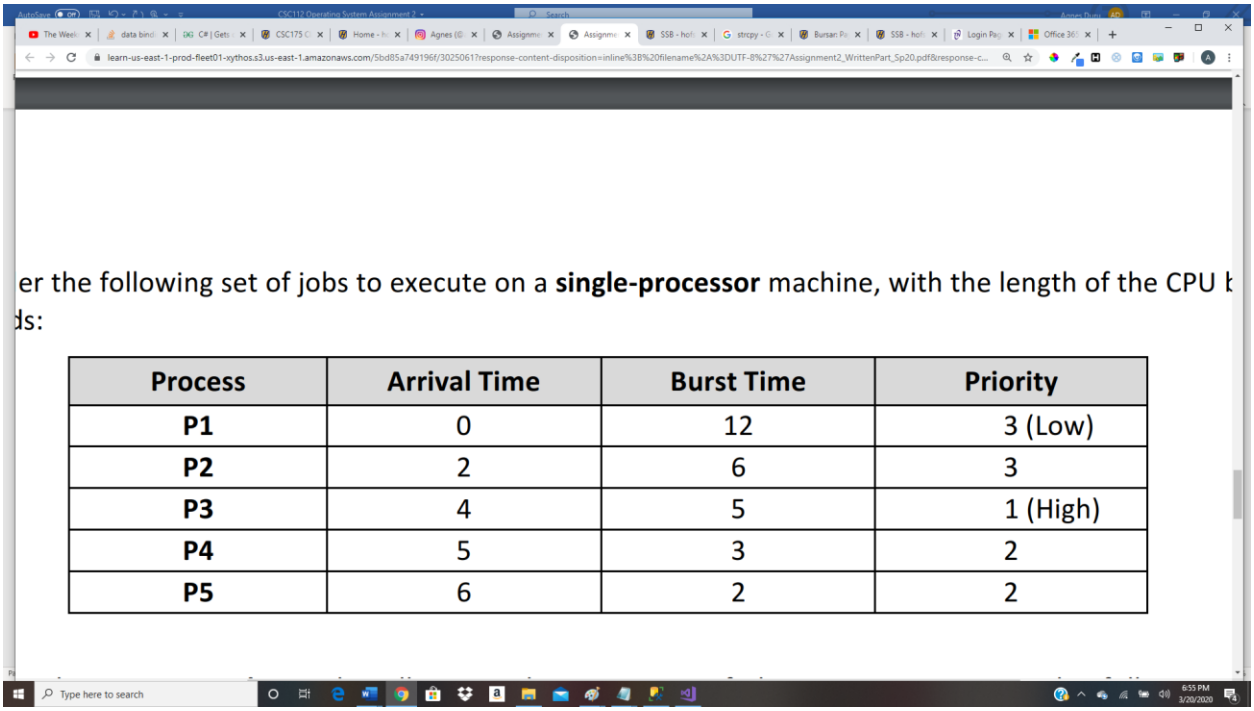
Output: Option B with the thread t2 being created after the print statement
Child A will print out → C: 10, Hello
Child A executes EXECV and gets list of argument/option values.
Parent will print→ D: 10, Hello
→A: 21 Goodbye
Child B will print→ E: 21, Goodbye
F: 51, Goodbye
B: 51, Goodbye

The guarantees regarding the final output are that 1. The parent process would not need to wait for child A because it went into the EXECV function. 2. The parent process calls on pthread join t1 and so that thread needs to finish before the main thread. 3. The program would immediately stop execution after func2 is executed because it calls on the exit(0) function. The possible first line of output is " C: 10, Hello"
. The possible last line of outputs are either "B: 51, Goodbye" or "F: 51, Goodbye".

v) What will change if we remove the line "wait(NULL)" in the code above? vi) What will change if in the code of func1 we replace the line "return NULL" with "exit(0)" ?

a. Nothing will change if we remove the line wait(Null) in the code above because the Child A processes called on the EXECV function and so the parent did not have a child to wait on. If we replace the line return NULL with exit(0) then the program would completely stop execution after returning the value from func1.

Question #3



er the following set of jobs to execute on a **single-processor** machine, with the length of the CPU b
ds:

Process	Arrival Time	Burst Time	Priority
P1	0	12	3 (Low)
P2	2	6	3
P3	4	5	1 (High)
P4	5	3	2
P5	6	2	2

ii. Draw the six Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, SRTF, RR (quantum = 3), and **preemptive and non-preemptive priority scheduling** sNote: For RR, if a new process Pj arrives at the same time that the time slice of the executing process Pi expires, the OS puts the executing process Pi to the ready queue, followed by the new process Pj. Note: Lower priority numbers indicate higher priority in the system

- FCFS : First in First Out. Add the next burst time amount to the prev one

P1	P2	P3	P4	P5

0 12 18 23 26 28

- SJF: Shortest Job First: P1 arrives first .[Get the one that arrived first and choose the ones with the shorter burs time next fcfs if same amount]

P1	P5	P4	P3	P2

0 12 14 17 22 28

- SRTF : Shortest Remaining Time First. If new process arrives at that segment unit w/ shorter CPU burst than the remaining for current process, schedule new process. Start with first process and if a new one comes and the burst is shorter subtract what was done for the prev process. Each time before scheduling new process check that the new one is the shortest then the remaining previous ones.

Process	Arrival Time	Original Burst	Remaining Burst
P1	0	12	12-2 ->10
P2	2	6	=0
P3	4	5	5->0
P4	5	3	3->0
P5	6	2	2->0

Process 1 starts from 0 to 2. CPU Burst is now 10. Process 2 arrives with a CPU burst of 6. 6<10. Process 2 starts
Process 2 starts from 2 to 4. CPU Burst is now 4. Process 3 arrives with a CPU burst of 5. 5>4 and so Process 2 finishes
Now all have arrived at this point.
Process 3 arrived with a CPU burst of 5.
Process 4 arrives with a CPU burst of 3.
Process 5 arrived with a CPU burst of 2.
Process 5 CPU 2< Process 4 CPU 3< Process 3 CPU 5<Process 1 CPU 10

P1	P2	P5	P4	P3	P1	
2	6	2	3	5	10	

0 2 8 10 13 18 28

Process	Arrival Time	Original Burst	Remaining Burst
P1	0	12	12-3 = 9-3=6-3=0
P2	2	6	6-3 =3-3=0 → 0
P3	4	5	5-3 =2 →0
P4	5	3	3-3=0
P5	6	2	0

- RR – Quantum = 3

P1	P2	P1	P3	P4	P2
3	3	3	3	3	3

0 3 6 9 12 15 →18

P5	P1	P3	P1
2	3	2	3

18 20 23 25 28

- Preemptive : Can interrupt

Process	Arrival Time	Original Burst	Priority	Remaining Burst
P1	0	12	3	12-4 =8
P2	2	6	3	
P3	4	5	1	=0
P4	5	3	2	
P5	6	2	2	

Process 1 has arrived. Process 2 arrives at unit 2. Has same priority so 1 continues. Process 2 is in waiting queue.

At 4 Process 3 arrives. It has a higher priority and interrupts process 1.

By now all processes have arrived.

Do Process 4,5,1 and finish 2

P1	P3	P4	P5	P1	P2	
4	5	3	2	8	6	

0 4 9 12 14 22 28

- Non-Preemptive: Cannot interrupt A process with the shortest burst time will execute first;

Process	Arrival Time	Original Burst	Priority	Remaining Burst
P1	0	12	3	
P2	2	6	3	
P3	4	5	1	
P4	5	3	2	
P5	6	2	2	

Process 1 finishes because it arrived first. Every process has arrived. P3 has a higher priority. Then P4,P5, Then P2

P1	P3	P4	P5	P2	
12	5	3	2	6	

0 12 17 20 22 28

i. ii. What is the per-process waiting time for each of these scheduling algorithms?

1. Waiting time = turnaround time-Burst time

- FCFS

Process	Waiting Time
P1	(12-12)=0
P2	(16-6) = 10
P3	(19-5) =14
P4	(21-3)=18
P5	(22-2) = 20

- SJF

Process	Waiting Time
P1	(12-12) = 0
P2	(26-6) = 20
P3	(18-5) =13
P4	(12-3) = 9
P5	(8-2) = 6

- SRTF

Process	Waiting Time
P1	(28-12) = 16
P2	(6-6) =0
P3	(14-5) = 9
P4	(8-3) = 5
P5	(4-2) = 2

- RR

Process	Waiting Time
P1	(28-12) = 16
P2	(16-6) = 10
P3	(21-5) = 16
P4	(10-3) = 7
P5	(14-2) = 12

- Preemptive Priority

Process	Waiting Time
P1	(22-12) = 10
P2	(26-6) = 20
P3	(5-5) = 0
P4	(7-3) = 4
P5	(8-2) = 6

- Non-Preemptive Priority

Process	Waiting Time
P1	(12-12) = 0
P2	(26-6) = 20
P3	(13-5) = 8
P4	(15-3) = 12
P5	(16-2) = 14

2. Turnaround time = Completed time – Arrival time
   - FCFS

Process	Turnaround Time
P1	(12-0)=12
P2	(18-2) = 16
P3	(23-4) =19
P4	(26-5) =21
P5	(28-6)= 22

   - SJF

Process	Turnaround Time
P1	(12-0) = 12
P2	(28-2) = 26
P3	(22-4) = 18
P4	(17-5) = 12
P5	(14-6) = 8

   - SRTF

Process	Turnaround Time
P1	(28-0) = 28
P2	(8-2) = 6
P3	(18-4) =14
P4	(13-5) = 8
P5	(10-6) = 4

   - RR

Process	Turnaround Time
P1	(28-0)= 28
P2	(18-2) = 16
P3	(25-4) = 21
P4	(15-5) = 10

P5	(20-6) = 14

- Preemptive Priority

Process	Turnaround Time
P1	(22-0) =22
P2	(28-2) = 26
P3	(9-4) = 5
P4	(12-5) = 7
P5	(14-6) = 8

- Non -Preemptive Priority

Process	Turnaround Time
P1	(12-0) = 12
P2	(28-2) = 26
P3	(17-4) = 13
P4	(20-5) = 15
P5	(22-6) = 16

iv. Which of these scheduling algorithm is a better choice for interactive processes? To justify your answer (and get full points) show the computations that led to your answer.

- Answer: Interactive process is I/O-bound (Interactive) task: mostly short CPU burst. {The Response time lowest}. FCFS wait time total is 62 , SJF wait time total is 48, SRTF wait time total is , RR wait time total is 32, Preemptive priority wait time total is 40, non-preemptive wait time total is 54. Round Robbin has the shortest wait time showing that it had the fasted response time. Round Robbin is the better choice for interactive processes.

v. Assuming that the time that it takes to switch between processes is not negligible, which is the most efficient algorithm (of the ones above) when the goal is to maximize the CPU utilization? To justify your answer (and get full points) show the computations that led to your answer.

- Answer: CPU- bound task: Mostly long CPU bursts {The least} FCFS has 4 context switches , SJF has 4 context switches, SRTF has 5 context switches ,RR has 8 context switches, Preemptive priority has 5 context switches, non-preemptive priority has 4 context switches. I found this by counting the amount of vertical lines right before every process. FCFS, SJF, non-preemptive have the least amount of context switches with that being 4.