Discussion 2/15/19

Quiz Review

Name two items in a PTE that are **not** found in a TTE.

Quiz Review

Name two items in a PTE that are **not** found in a TTE.

CPU status word

Stack pointer

Process state

Priority / scheduling parameters

Process ID

Parent process ID

Signals

Process start time

Total CPU usage

Quiz Review

Name four events that could cause scheduling to occur.

Quiz Review

Name four events that could cause scheduling to occur.

- Process Creation
- Process Exit
- Blocked
- I/O Interrupt
- Clock Interrupts

Quiz Review

Name a pro and con for both micro and macro kernels

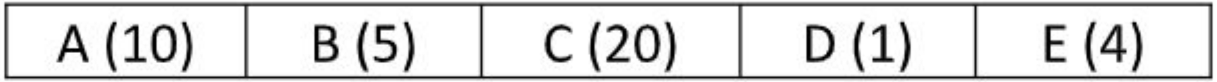
Quiz Review

Name a pro and con for both micro and macro kernels

- Pros
  - Micro: Small size in memory!
  - Macro: Faster because modules are all in same address space, less context switches
- Cons
  - Micro: Slower because more switches, modules are in different address spaces!
  - Macro: A bug in one module could cause the whole system to fail!

Quiz Review

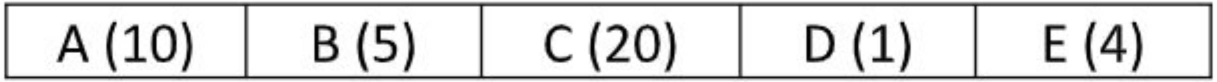
Given SJF scheduling and all processes arriving at time 0, what is the average turnaround time and throughput?



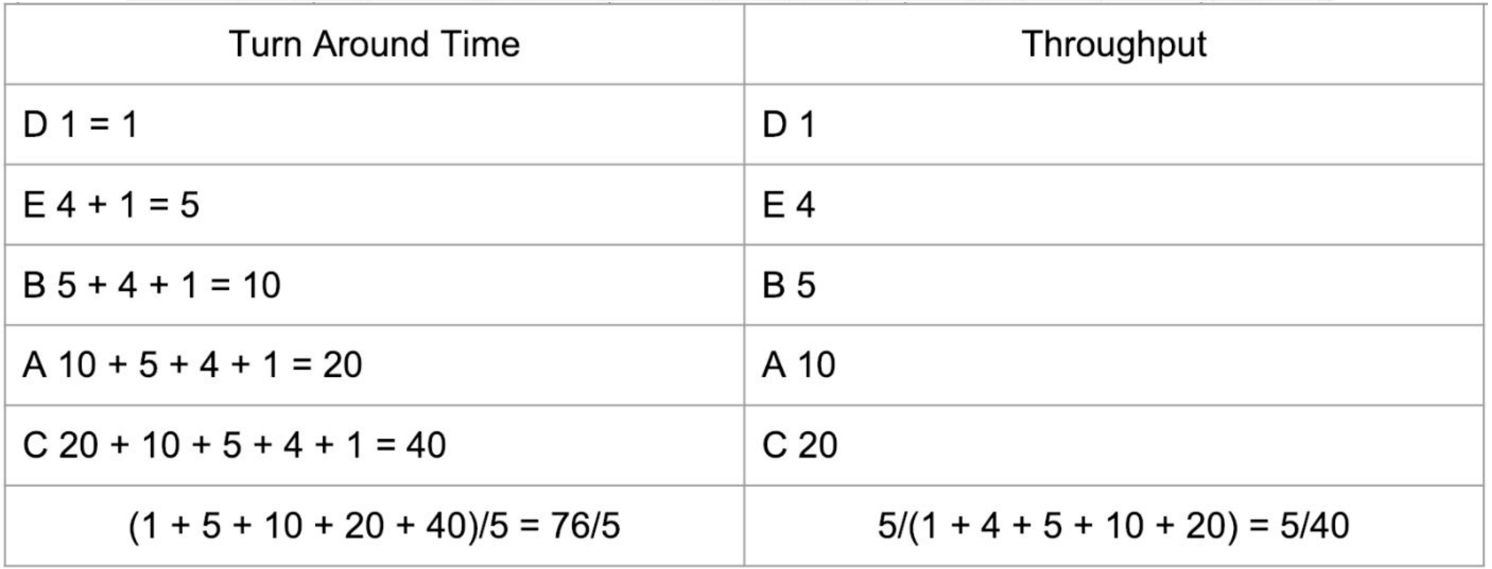
A (10)	B (5)	C (20)	D (1)	E (4)

Quiz Review

Given SJF scheduling and all processes arriving at time 0, what is the average turnaround time and throughput?



A (10)	B (5)	C (20)	D (1)	E (4)



Turn Around Time	Throughput
D 1 = 1	D 1
E 4 + 1 = 5	E 4
B 5 + 4 + 1 = 10	B 5
A 10 + 5 + 4 + 1 = 20	A 10
C 20 + 10 + 5 + 4 + 1 = 40	C 20
(1 + 5 + 10 + 20 + 40)/5 = 76/5	5/(1 + 4 + 5 + 10 + 20) = 5/40

Scheduling

Consider three CPU-intensive processes, which require 30, 20, and 10 time units and arrive at times 0, 2 and 6, respectively.

- How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.
- What is the average turnaround time?
- What is the throughput?

Scheduling

Consider three CPU-intensive processes, which require 10, 20, 30, and 20 time units and arrive at times 0, 2, 6, and 12, respectively.

- How many context switches are needed if the operating system implements a round robin scheduling algorithm with quantum of 5? Do not count the context switches at time zero and at the end.
- What is the average turnaround time?
- What is the throughput?

Scheduling

Consider three CPU-intensive processes, which require 10, 20, 30, and time units and all arrive at time 0, 2, and 6 respectively.

- What happens under guaranteed scheduling with a quantum of 5?

Scheduling

Which of the following process scheduling algorithm may lead to starvation:

- FIFO
- Round Robin
- Shortest Job Next
- None of the above

Race Conditions

- An execution ordering of concurrent flows that results in undesired behavior is called a race condition—a software defect and frequent source of vulnerabilities.
- Race conditions result from runtime environments, including operating systems, that must control access to shared resources, especially through process scheduling.
- Take a minute and come up with an example of a race condition.
  - Remember:
    - `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);`
    - `int pthread_join(pthread_t thread, void **retval);`
    - `void pthread_exit(void *retval);`

Race Conditions

```
int arr[5];

void *set_params(void *args) {
    for (int i = 0; i < 5; i++) {
        arr[i] = *((int *)args);
    }
    print(arr);
}
```

```
void main() {
    int x = 0, y = 5;
    pthread_t t1, t2;
    pthread_create(&t1, -1, set_params, &x);
    pthread_create(&t2, -1, set_params, &y);
    pthread_join(t1);
    pthread_join(t2);
}
```

Ways to Avoid Race Conditions

- Mutex
- Strict Altercation

- Fix your race conditions given:
    - pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
    - pthread_mutex_lock(&mutex);
    - pthread_mutex_unlock(&mutex);

Ways to Avoid Race Conditions

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int arr[5];
void *set_params(void *args) {
    pthread_mutex_lock(mutex);
    for (int i = 0; i < 5; i++) {
        arr[i] = *((int *)args);
    }
    print(arr);
    pthread_mutex_unlock(mutex);
}
```

```
void main() {
    int x = 0, y = 5;
    pthread_t t1, t2;
    pthread_create(&t1, -1, set_params, &x);
    pthread_create(&t2, -1, set_params, &y);
    pthread_join(t1);
    pthread_join(t2);
}
```