دانشکده مهندسی کامپیوتر

درس سیستم‌های عامل

پاسخنامه کوییز اول

مدرس ..... **دکتر رضا انتظاری ملکی**

طراح ..... **فرزان رحمانی**

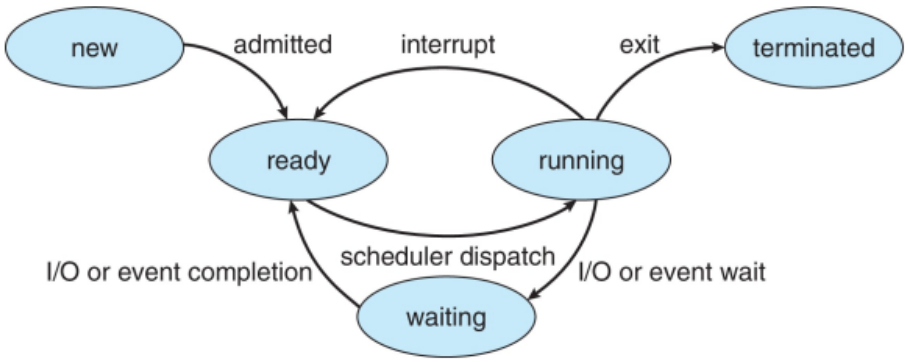
۱) با کشیدن نمودار حالت های مختلف process (Diagram of Process State) هر یک از حالت های آن را توضیح دهید.

پاسخ:

Process State

- As a process executes, it changes **state**
  - **New**: The process is being created
  - **Running**: Instructions are being executed
  - **Waiting**: The process is waiting for some event to occur
  - **Ready**: The process is waiting to be assigned to a processor
  - **Terminated**: The process has finished execution

Diagram of Process State



۲) در این سوال باید دنباله π را با استفاده از Thread محاسبه کنید. برنامه زیر را کامل کنید تا با استفاده از فرمول زیر دنباله π را تا تعداد معینی عبارت محاسبه کند.

$$\sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

به عنوان مثال، اگر شما تا 10 میلیونیم این دنباله را در برنامه خود با استفاده از 10 رشته محاسبه کنید، هر رشته باید یک میلیون عبارت از این دنباله را محاسبه کند. قسمت های زیر ... ? // را کامل کنید.

پاسخ:

```
// ? get the arguments; start and end
int start = args->start; // (*args).start;
int end = args->end; // (*args).end;

// ? initialize pi
double pi = 0.0;

// ? add to pi suitable value
pi += 1.0 / (2 * i + 1);


// ? add to pi suitable value
pi -= 1.0 / (2 * i + 1);


// ? set return value
rvals->ret = pi;


// ? calculate the remainder
int remainder = num_of_terms % num_of_threads;


// ? create threads pointers using malloc
pthread_t *threads = malloc(sizeof(pthread_t) * num_of_threads);
```

```
// ? create thread and pass the arguments to the thread
pthread_create(&threads[i], NULL, calculate_partial_pi, &args[i]);

// ? wait for each thread to finish and save the return values in rvals
pthread_join(threads[i], (void **) &rvals);
```

کل کد:

```
#include "stdlib.h"
#include "stdio.h"
#include "unistd.h"
#include "string.h"
#include "pthread.h"
#include <assert.h>

typedef struct {int start; int end;} myarg_t;
typedef struct {double ret;} myret_t;

void *calculate_partial_pi(void *arg)
{
    myarg_t *args = (myarg_t *) arg;
    // ? get the arguments; start and end
    int start = args->start; // (*args).start;
    int end = args->end; // (*args).end;
    // ? initialize pi
    double pi = 0.0;
    for (int i = start; i < end; i++)
    {
        if (i % 2 == 0)
        {
            // ? add to pi suitable value
            pi += 1.0 / (2 * i + 1);
        }
        else
        {
            // ? add to pi suitable value
            pi -= 1.0 / (2 * i + 1);
        }
    }
    myret_t *rvals = malloc(sizeof(myret_t));
    assert(rvals != NULL);  // if conditon is True => return error
```

```
    // ? set return value
    rvals->ret = pi;
    return (void *) rvals;
}

int main(int argc, char *argv[])
{
    // calulate pi using series formula
    int num_of_terms = 0;
    int num_of_threads = 0;
    printf("Enter the number of terms: ");
    scanf("%d", &num_of_terms);
    printf("Enter the number of threads: ");
    scanf("%d", &num_of_threads);
    assert(num_of_terms > 0);
    assert(num_of_threads > 0);
    assert(num_of_terms >= num_of_threads);

    int terms_per_thread = num_of_terms / num_of_threads;
    // ? calculate the remainder
    int remainder = num_of_terms % num_of_threads;
    double pi = 0.0;

    // ? create threads pointers using malloc
    pthread_t *threads = malloc(sizeof(pthread_t) * num_of_threads);
    assert(threads != NULL);
    myarg_t *args = malloc(sizeof(myarg_t) * num_of_threads);
    assert(args != NULL);
    for (int i = 0; i < num_of_threads; i++)
    {
        args[i].start = i * terms_per_thread;
        args[i].end = (i + 1) * terms_per_thread;
        if (i == num_of_threads - 1) // last thread
        {
            args[i].end += remainder;
        }
        // ? create thread and pass the arguments to the thread
        pthread_create(&threads[i], NULL, calculate_partial_pi, &args[i]);
    }

    myret_t *rvals; // return values of thread
    for (int i = 0; i < num_of_threads; i++)
    {
```

```
        // ? wait for each thread to finish and save the return values in rvals
        pthread_join(threads[i], (void **) &rvals);
        pi += rvals->ret; // return value of thread
        free(rvals);
    }

    pi *= 4.0;
    printf("pi = %.10f\n", pi);
    free(threads);
    free(args);
    return 0;
}
```