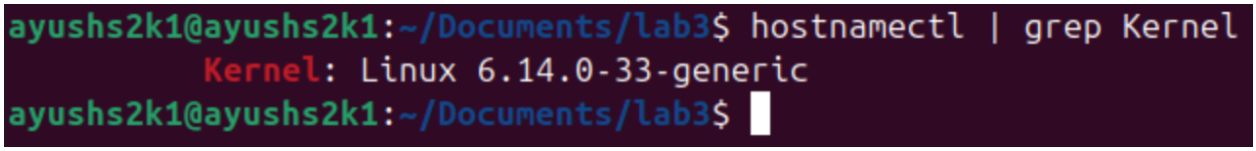
Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

New York University
Tandon School of Engineering
Department of Computer Science and Engineering
Introduction to Operating Systems
Fall 2025
Assignment 6

Kernel Version: Linux 6.14.0-33-generic



Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

PART A

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <errno.h>
#include <sys/wait.h>

#define PORT 8080
#define BUFFER_SIZE 256


int main(){
   pid_t pid;
   int sockfd, newsockfd;
   struct sockaddr_in server_addr, client_addr;
   socklen_t client_len;

   pid = fork();

   if(pid < 0){
       perror("Fork Failed!");
       exit(1);
   }

   if(pid > 0){
       // Parent Process - Server (Consumer)
       printf("[Server] Parent process started (PID: %d)\n", getpid());

       // Insert random wait (0 to 6999 ms) before starting to listen
       srand(time(NULL));
       int wait_time = rand() % 7000;
       printf("[Server] Waiting %d ms before starting...\n", wait_time);
       usleep(wait_time * 1000);

       // Creating socket
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

```
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0){
        perror("Error opening socket!");
        exit(1);
    }

    // Allow socket reuse to avoid "Address already in use" errors
    int opt = 1;
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    // Initialize server address structure
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind socket to port
    if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
        perror("Error on binding!");
        close(sockfd);
        exit(1);
    }

    // Listen for connections (backlog of 5)
    listen(sockfd, 5);
    printf("[Server] Listening on port %d...\n", PORT);

    // Accept Connection from client
    client_len = sizeof(client_addr);
    newsockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);
    if (newsockfd < 0){
        perror("Error on accepting connection from client!");
        close(sockfd);
        exit(1);
    }

    printf("[Server] Connection accepted from client\n");

    // Read messages from client
    char buffer[BUFFER_SIZE];
    int n;
    int message_count = 0;
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

```
    while (1){
        memset(buffer, 0, BUFFER_SIZE);
        n = read(newsockfd, buffer, BUFFER_SIZE - 1);

        if (n < 0){
            perror("Error reading from socket!");
            break;
        }
        else if (n == 0){
            printf("[Server] Client closed connection\n");
            break;
        }

        printf("[Server] Received message %d\n", ++message_count);

        // Check for termination message
        if (strncmp(buffer, "END", 3) == 0){
            printf("[Server] Received END signal, shutting down\n");
            break;
        }

        // Send acknowledgment back to client
        char ack[BUFFER_SIZE];
        snprintf(ack, BUFFER_SIZE, "ACK %d", message_count);
        write(newsockfd, ack, strlen(ack));
    }

    // Close sockets
    close(newsockfd);
    close(sockfd);

    // Wait for child process to finish
    wait(NULL);
    printf("[Server] Server terminated\n");
}
else{
    // Child process - Client (Producer)
    printf("[Client] Child process started (PID: %d)\n", getpid());

    // Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

```
        if (sockfd < 0){
            perror("Error opening socket!");
            exit(1);
        }

        // Initialize server address
        memset(&server_addr, 0, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(PORT);

        // Convert IPv4 address from text to binary
        if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0){
            perror("Invalid address!");
            close(sockfd);
            exit(1);
        }

        // Repeatedly attempt to connect until successful
        printf("[Client] Attempting to connect to server...\n");
        int connected = 0;
        int attempts = 0;

        while (!connected){
            attempts++;
            if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
0){
                if (errno == ECONNREFUSED){
                    // Server not ready, wait and retry
                    printf("[Client] Connection refused, retrying... (attempt %d)\n",
attempts);
                    usleep(100000); // Wait 100 ms
                }
                else{
                    perror("Connection error!");
                    close(sockfd);
                    exit(1);
                }
            }
            else{
                connected = 1;
                printf("[Client] Connected to server after %d attempts\n", attempts);
            }
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

```
        }

        // Send messages to server
        char buffer[BUFFER_SIZE];
        for (int i = 1; i <= 5; i++){
            snprintf(buffer, BUFFER_SIZE, "Message %d from producer", i);
            printf("[Client] Sending: %s\n", buffer);
            write(sockfd, buffer, strlen(buffer));

            // Read acknowledgment
            memset(buffer, 0, BUFFER_SIZE);
            read(sockfd, buffer, BUFFER_SIZE - 1);
            printf("[Client] Received: %s\n", buffer);

            usleep(500000); // Wait 500 ms between messages
        }

        // Send termination message
        strcpy(buffer, "END");
        printf("[Client] Sending: %s\n", buffer);
        write(sockfd, buffer, strlen(buffer));

        // Close socket
        close(sockfd);
        printf("[Client] Client terminated\n");
        exit(0);
    }
    return 0;
}
```

Command used to compile the program: gcc -o lab6_a lab6_a.c -Wall
After compiling, it creates an executable file, lab6_a, as shown below.

```
ayushs2k1@ayushs2k1:~/Documents/lab6$ ls -lrt
total 8
-rw-rw-r-- 1 ayushs2k1 ayushs2k1 5776 Nov  9 15:02 lab6_a.c
ayushs2k1@ayushs2k1:~/Documents/lab6$ gcc -o lab6_a lab6_a.c -Wall
ayushs2k1@ayushs2k1:~/Documents/lab6$ ls -lrt
total 28
-rw-rw-r-- 1 ayushs2k1 ayushs2k1  5776 Nov  9 15:02 lab6_a.c
-rwxrwxr-x 1 ayushs2k1 ayushs2k1 71576 Nov  9 15:02 lab6_a
ayushs2k1@ayushs2k1:~/Documents/lab6$
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

Running the executable:

```
ayushs2k1@ayushs2k1:~/Documents/lab6$ ./lab6_a
[Server] Parent process started (PID: 20861)
[Server] Waiting 1673 ms before starting...
[Client] Child process started (PID: 20862)
[Client] Attempting to connect to server...
[Client] Connection refused, retrying... (attempt 1)
[Client] Connection refused, retrying... (attempt 2)
[Client] Connection refused, retrying... (attempt 3)
[Client] Connection refused, retrying... (attempt 4)
[Client] Connection refused, retrying... (attempt 5)
[Client] Connection refused, retrying... (attempt 6)
[Client] Connection refused, retrying... (attempt 7)
[Client] Connection refused, retrying... (attempt 8)
[Client] Connection refused, retrying... (attempt 9)
[Client] Connection refused, retrying... (attempt 10)
[Client] Connection refused, retrying... (attempt 11)
[Client] Connection refused, retrying... (attempt 12)
[Client] Connection refused, retrying... (attempt 13)
[Client] Connection refused, retrying... (attempt 14)
[Client] Connection refused, retrying... (attempt 15)
[Client] Connection refused, retrying... (attempt 16)
[Server] Listening on port 8080...
[Client] Connected to server after 17 attempts
[Client] Sending: Message 1 from producer
[Server] Connection accepted from client
[Server] Received message 1
[Client] Received: ACK 1
[Client] Sending: Message 2 from producer
[Server] Received message 2
[Client] Received: ACK 2
[Client] Sending: Message 3 from producer
[Server] Received message 3
[Client] Received: ACK 3
[Client] Sending: Message 4 from producer
[Server] Received message 4
[Client] Received: ACK 4
[Client] Sending: Message 5 from producer
[Server] Received message 5
[Client] Received: ACK 5
[Client] Sending: END
[Client] Client terminated
[Server] Received message 6
[Server] Received END signal, shutting down
[Server] Server terminated
ayushs2k1@ayushs2k1:~/Documents/lab6$
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

Which of the calls above are blocking and which are not?

Blocking calls are:

- accept() - Blocks until a client connects
- read() - Blocks until data is available to read
- write() - Can block if the send buffer is full
- connect() - Blocks until the connection is established or fails

Non-blocking calls are:

- socket() - Returns immediately with a socket descriptor
- bind() - Returns immediately after binding
- listen() - Returns immediately after setting up the listen queue
- close() - Returns immediately after closing the socket

Is this a form of direct communication or indirect communication?

This is a form of **direct communication**. The client explicitly connects to the server using the server's address and port number. Both processes must be aware of each other for communication to occur. The client knows the server's IP address (127.0.0.1) and port (8080), and once connected, they communicate directly through the established socket connection.

What is the failure flag returned from connect() that indicates the server is not ready?

The failure flag is ECONNREFUSED, which indicates the server is not ready.

How would you change your program to communicate between processes on different machines?

To communicate between different machines, we need to replace "127.0.0.1" (localhost) with the actual IP address of the remote server machine.

```
inet_pton(AF_INET, "<IP_ADDRESS>", &server_addr.sin_addr)
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

PART B

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <math.h>
#include <time.h>

#define NUM_THREADS 4
#define NUM_POINTS 1000000


// Shared variable for counting points inside circle
int count =0;

// Semaphore to protect critical section
sem_t semaphore;


// Worker Thread function
void *WorkerThread(void *arg){
   // Seed random number generator with unique value for each thread
   unsigned int rand_state = (unsigned int) time(NULL) + pthread_self();

   // Generate random points
   for(int i=0; i<NUM_POINTS; i++){
       // Generate random x and y coordinates between -1 and 1
       double x = (double)rand_r(&rand_state) / RAND_MAX * 2.0 - 1.0;
       double y = (double)rand_r(&rand_state) / RAND_MAX * 2.0 - 1.0;

       // Calculate radius
       double r = sqrt(x * x + y * y);

       // Check if point is inside the circle
       if(r <= 1.0){
           // Enter Critical Section
           sem_wait(&semaphore);
           count++;
           sem_post(&semaphore);
           // Exit Critical Section
       }
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

```
    }

    printf("Thread %lu completed. Generated %d points.\n", pthread_self(), NUM_POINTS);
    pthread_exit(NULL);
}

int main(){
    pthread_t threads[NUM_THREADS];
    int rc;

    printf("Monte Carlo Simulation to Estimate Pi\n");
    printf("=====================================\n");
    printf("Number of threads: %d\n", NUM_THREADS);
    printf("Points per thread: %d\n", NUM_POINTS);
    printf("Total points: %d\n", NUM_THREADS * NUM_POINTS);
    printf("\n");


    // Initialize semaphore (binary semaphore with initial value as 1)
    if(sem_init(&semaphore, 0, 1) !=0){
        perror("Semaphore initialization failed!");
        exit(1);
    }

    // Create worker threads
    printf("Creating %d worker threads...\n", NUM_THREADS);
    for(int i=0; i<NUM_THREADS; i++){
        rc = pthread_create(&threads[i], NULL, WorkerThread, NULL);
        if(rc){
            printf("Error: Unable to create thread %d, return code %d\n", i, rc);
            exit(1);
        }
    }

    // Wait for all threads to complete
    printf("\nWaiting for threads to complete...\n");
    for(int i=0; i<NUM_THREADS; i++){
        pthread_join(threads[i], NULL);
    }

    // Calculate and print the estimate value of pi
    int total_points = NUM_THREADS * NUM_POINTS;
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

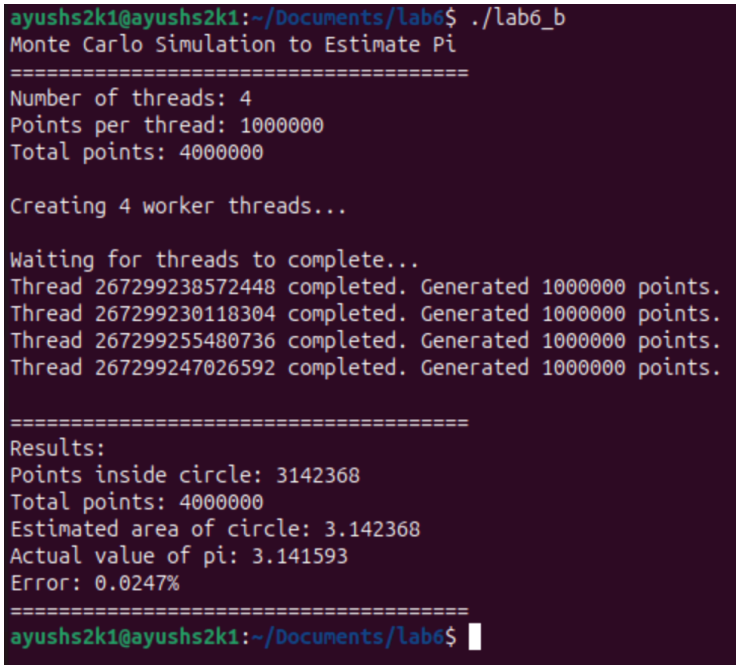
```c
    double pi_estimate = 4.0 * (double)count / (double)total_points;

    printf("\n========================================\n");
    printf("Results:\n");
    printf("Points inside circle: %d\n", count);
    printf("Total points: %d\n", total_points);
    printf("Estimated area of circle: %.6f\n", pi_estimate);
    printf("Actual value of pi: %.6f\n", M_PI);
    printf("Error: %.4f%%\n", fabs(pi_estimate - M_PI) / M_PI * 100);
    printf("========================================\n");

    // Destroy Semaphore
    sem_destroy(&semaphore);

    return 0;
}
```

Command used to compile the program: gcc -o lab6_b lab6_b.c -pthread -lm -Wall
After compiling, it creates an executable file, lab6_b, as shown below.

```
ayushs2k1@ayushs2k1:~/Documents/lab6$ ls -lrt
total 32
-rw-rw-r-- 1 ayushs2k1 ayushs2k1  5776 Nov  9 15:02 lab6_a.c
-rwxrwxr-x 1 ayushs2k1 ayushs2k1 71576 Nov  9 15:02 lab6_a
-rw-rw-r-- 1 ayushs2k1 ayushs2k1  2712 Nov  9 18:23 lab6_b.c
ayushs2k1@ayushs2k1:~/Documents/lab6$ gcc -o lab6_b lab6_b.c -pthread -lm -Wall
ayushs2k1@ayushs2k1:~/Documents/lab6$ ls -lrt
total 52
-rw-rw-r-- 1 ayushs2k1 ayushs2k1  5776 Nov  9 15:02 lab6_a.c
-rwxrwxr-x 1 ayushs2k1 ayushs2k1 71576 Nov  9 15:02 lab6_a
-rw-rw-r-- 1 ayushs2k1 ayushs2k1  2712 Nov  9 18:23 lab6_b.c
-rwxrwxr-x 1 ayushs2k1 ayushs2k1 71288 Nov  9 18:25 lab6_b
ayushs2k1@ayushs2k1:~/Documents/lab6$
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

Running the executable:



Which of the calls above are blocking and which are not?

Blocking calls are:

- sem_wait() - Blocks if the semaphore value is 0, waits until it becomes available
- pthread_join() - Blocks until the specified thread terminates

Non-blocking calls are:

- sem_init() - Returns immediately after initializing the semaphore
- sem_post() - Returns immediately after incrementing the semaphore
- pthread_create() - Returns immediately after creating the thread
- sem_destroy() - Returns immediately after destroying the semaphore

Is this a form of direct communication or indirect communication?

This is a form of **indirect communication**. The threads communicate through a shared variable (count) protected by a semaphore. They don't directly pass messages to each other; instead, they coordinate access to shared memory. The semaphore acts as a synchronization mechanism to ensure mutual exclusion when accessing the critical section.

-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-**END**-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x