

Module 2 Assignment

Task 1: C Program with 3 Threads (Prime sum, periodic messages)

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

int is_prime(int num) {
    if (num <= 1) return 0;
    for (int i = 2; i * i <= num; i++)
        if (num % i == 0) return 0;
    return 1;
}

void* threadA(void* arg) {
    int N = *(int*)arg;
    int count = 0, num = 2, sum = 0;

    while (count < N) {
        if (is_prime(num)) {
            sum += num;
            count++;
        }
        num++;
    }

    printf("Thread A: Sum of first %d prime numbers is %d\n", N, sum);
    pthread_exit(NULL);
}

void* threadB(void* arg) {
    int elapsed = 0;
    while (elapsed < 100) {
```

```

        printf("Thread 1 running\n");
        sleep(2);
        elapsed += 2;
    }
    pthread_exit(NULL);
}

void* threadC(void* arg) {
    int elapsed = 0;
    while (elapsed < 100) {
        printf("Thread 2 running\n");
        sleep(3);
        elapsed += 3;
    }
    pthread_exit(NULL);
}

int main() {
    int N;
    printf("Enter the value of N: ");
    scanf("%d", &N);
    pthread_t t1, t2, t3;
    pthread_create(&t1, NULL, threadA, &N);
    pthread_create(&t2, NULL, threadB, NULL);
    pthread_create(&t3, NULL, threadC, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    printf("All threads completed.\n");
    return 0;
}

```

Code Snippet:

```

1 // On this question
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <pthread.h>
5 #include <unistd.h>
6
7 int is_prime(int num) {
8     if (num <= 1) return 0;
9     for (int i = 2; i * i <= num; i++)
10         if (num % i == 0) return 0;
11     return 1;
12 }
13
14 void* threadA(void* arg) {
15     int N = *(int*)arg;
16     int count = 0, num = 2, sum = 0;
17
18     while (count < N) {
19         if (is_prime(num)) {
20             sum += num;
21             count++;
22         }
23         num++;
24     }
25
26     printf("Thread A: Sum of first %d prime numbers is %d\n", N, sum);
27     pthread_exit(NULL);
28 }
29
30 void* threadB(void* arg) {
31     int elapsed = 0;
32     while (elapsed < 100) {
33         printf("Thread 1 running\n");
34         sleep(2);
35         elapsed += 2;
36     }
37     pthread_exit(NULL);
38 }
39
40 void* threadC(void* arg) {
41     int elapsed = 0;
42     while (elapsed < 100) {
43         printf("Thread 2 running\n");
44         sleep(3);
45         elapsed += 3;
46     }
47     pthread_exit(NULL);
48 }
49
50 int main() {
51     int N;
52     printf("Enter the value of N: ");
53     scanf("%d", &N);
54
55     pthread_t t1, t2, t3;
56
57     pthread_create(&t1, NULL, threadA, &N);
58     pthread_create(&t2, NULL, threadB, NULL);
59     pthread_create(&t3, NULL, threadC, NULL);
60
61     pthread_join(t1, NULL);
62     pthread_join(t2, NULL);
63     pthread_join(t3, NULL);
64
65     printf("All threads completed.\n");
66     return 0;
67 }

```

Output Snippet:

[illegible]

Task 2: C Program with 3 Threads (Prime sum, periodic messages) + SIGINT

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>

volatile sig_atomic_t keep_running = 1;

void handle_sigint(int sig) {
    printf("\nSIGINT received. Ignoring termination...\n");
    keep_running = 0;
}

int is_prime(int num) {
    if (num <= 1) return 0;
    for (int i = 2; i * i <= num; i++)
        if (num % i == 0) return 0;
    return 1;
}

void* threadA(void* arg) {
    int N = *(int*)arg;
    int count = 0, num = 2, sum = 0;
    clock_t start = clock();

    while (count < N) {
        if (is_prime(num)) {
```

```

        sum += num;

        count++;
    }
    num++;
}

clock_t end = clock();
double time_taken = (double)(end - start) / CLOCKS_PER_SEC;

printf("Thread A: Sum of first %d prime numbers is %d (time: %.2fs)\n", N, sum,
time_taken);
pthread_exit(NULL);
}

void* threadB(void* arg) {
    int elapsed = 0;
    while (elapsed < 100 && keep_running) {
        printf("Thread 1 running\n");
        sleep(2);
        elapsed += 2;
    }
    pthread_exit(NULL);
}

void* threadC(void* arg) {
    int elapsed = 0;
    while (elapsed < 100 && keep_running) {
        printf("Thread 2 running\n");
        sleep(3);
        elapsed += 3;
    }
}

```

```
    pthread_exit(NULL);
}

int main() {
    signal(SIGINT, handle_sigint);

    int N;
    printf("Enter the value of N: ");
    scanf("%d", &N);

    pthread_t t1, t2, t3;

    pthread_create(&t1, NULL, threadA, &N);
    pthread_create(&t2, NULL, threadB, NULL);
    pthread_create(&t3, NULL, threadC, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    printf("All threads completed.\n");
    return 0;
}
```

Code Snippet:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <signal.h>
6 #include <time.h>
7
8 volatile sig_atomic_t keep_running = 1;
9 void handle_sigint(int sig) {
10     printf("\nSIGINT received. Ignoring termination...\n");
11     keep_running = 0;
12 }
13 int is_prime(int num) {
14     if (num <= 1) return 0;
15     for (int i = 2; i * i <= num; i++)
16         if (num % i == 0) return 0;
17     return 1;
18 }
19 void* threadA(void* arg) {
20     int N = *(int*)arg;
21     int count = 0, num = 2, sum = 0;
22     clock_t start = clock();
23
24     while (count < N) {
25         if (is_prime(num)) {
26             sum += num;
27             count++;
28         }
29         num++;
30     }
31
32     clock_t end = clock();
33     double time_taken = (double)(end - start) / CLOCKS_PER_SEC;
34
35     printf("Thread A: Sum of first %d prime numbers is %d (time: %.2fs)\n", N, sum, time_taken);
36     pthread_exit(NULL);
37 }
38 void* threadB(void* arg) {
39     int elapsed = 0;
40     while (elapsed < 100 && keep_running) {
41         printf("Thread 1 running\n");
42         sleep(2);
43         elapsed += 2;
44     }
45     pthread_exit(NULL);
46 }
47 void* threadC(void* arg) {
48     int elapsed = 0;
49     while (elapsed < 100 && keep_running) {
50         printf("Thread 2 running\n");
51         sleep(3);
52         elapsed += 3;
53     }
54     pthread_exit(NULL);
55 }
56 int main() {
57     signal(SIGINT, handle_sigint);
58     int N;
59     printf("Enter the value of N: ");
60     scanf("%d", &N);
61     pthread_t t1, t2, t3;
62     pthread_create(&t1, NULL, threadA, &N);
63     pthread_create(&t2, NULL, threadB, NULL);
64     pthread_create(&t3, NULL, threadC, NULL);
65     pthread_join(t1, NULL);
66     pthread_join(t2, NULL);
67     pthread_join(t3, NULL);
68     printf("All threads completed.\n");
69     return 0;
70 }
```

Output Snippet:

```
Enter the value of N: 10
Thread A: Sum of first 10 prime numbers is 129 (time: 0.00s)
Thread 1 running
Thread 2 running
Thread 1 running
Thread 2 running
Thread 1 running
Thread 2 running
Thread 1 running
Thread 2 running
Thread 1 running
Thread 2 running
Thread 1 running
```

Task 3: Notes on Topics

1. Child Process – fork()

The fork() system call in C is used to create a new process by duplicating the current process. The new process is called the child, and the original is the parent.

- Syntax: `pid_t pid = fork();`
- Return Values:
 - `> 0` → Parent process (returns child PID)
 - `= 0` → Child process
 - `< 0` → Error (process creation failed)

Example:

```
pid_t pid = fork();
if (pid == 0) {
    printf("Child process\n");
} else if (pid > 0) {
    printf("Parent process\n");
}
```

Use Cases: Creating background tasks, process pools, handling multitasking in UNIX.

2. Handling Common Signals

Signals are software interrupts delivered to a process to notify it of events.

- Common signals:
 - SIGINT: Interrupt from keyboard (Ctrl + C)
 - SIGTERM: Termination request
 - SIGKILL: Kill the process forcefully
 - SIGSEGV: Segmentation fault

Signal Handling in C:

```
#include <signal.h>

void handle_sigint(int sig) {
    printf("Caught SIGINT\n");
}
```



```
int main() {  
    signal(SIGINT, handle_sigint);  
    while (1);  
}
```

Use Cases: Clean exit, releasing memory, stopping threads gracefully.

3. Exploring Kernel Crashes

Kernel crashes are serious errors that occur in the core of an operating system.

Causes:

- Illegal memory access
- Bad device drivers
- Stack overflow in kernel space
- Hardware failure

How to Explore:

- Use dmesg to view kernel logs.
- Check /var/log/kern.log or /var/log/syslog
- Enable kernel crash dumps with kdump in Linux

Debugging Tools:

- gdb
- crash
- Kernel logging with printk()

Preventive Practice: Write safe kernel modules, avoid null pointer dereferences.

4. Time Complexity

Time complexity expresses the runtime growth of an algorithm in terms of input size N.

- Big O Notation Examples:
 - $O(1)$ – Constant (e.g., accessing an array element)
 - $O(n)$ – Linear (e.g., for loop over N elements)
 - $O(n^2)$ – Quadratic (e.g., nested loops)

Use Case in Thread A (Prime Calculation):

```
while (count < N) {  
    // Check if num is prime  
}
```

- Time Complexity: Roughly $O(N\sqrt{M})$ where M is the Nth prime candidate.
- Helps in optimizing code.
- Ensures scalability and better performance.

5. Locking Mechanism – mutex and spinlock

Feature	Mutex	Spinlock
Blocking	Yes	No (spins)
CPU Usage	Low	High
Context Switch	Yes	No
Best For	User space	Kernel / Low wait time

Locks are used to synchronize access to shared data in multi-threaded programs.

1. Mutex (Mutual Exclusion)

- Uses blocking mechanism. If a thread tries to lock a mutex that's already locked, it waits (sleeps).
- Safer for threads in user space.

Example:

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;  
pthread_mutex_lock(&lock);  
// critical section  
pthread_mutex_unlock(&lock);
```

2. Spinlock

- The thread actively waits in a loop (spins) until the lock is available.
- Used in kernel space or when wait time is expected to be very short.

Example:

```
#include <stdatomic.h>

atomic_flag lock = ATOMIC_FLAG_INIT;

while (atomic_flag_test_and_set(&lock)) {
    // spin
}

// critical section

atomic_flag_clear(&lock);
```