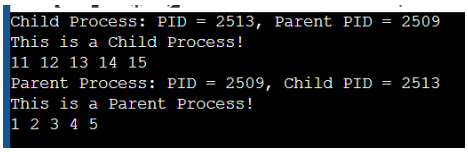
Lab Practice

Basic Parent and Child Processes Creation using FORK() command:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
    int array[5] = {1, 2, 3, 4, 5};

    pid_t pid = fork();

    if(pid == 0)
    {
        printf("Child Process: PID = %d, Parent PID = %d\n", getpid(), getppid());
        printf("This is a Child Process!\n");

        for (int i = 0; i < 5; i++)
        {
            array[i] += 10;
            printf("%d ", array[i]);
        }
        printf("\n");

        exit(0);
    }
    else
    {
        wait(NULL);

        printf("Parent Process: PID = %d, Child PID = %d\n", getpid(), pid);
        printf("This is a Parent Process!\n");

        for (int i = 0; i < 5; i++)
        {
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    return 0;
}
```



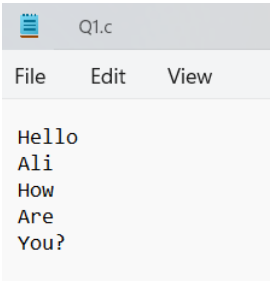
File Handling in C language:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
```

```c
#include <ctype.h>
#include <string.h>

int main(int argc, char* argv[])
{
    const char* filename = argv[1];

    pid_t pid = fork();

    if(pid == 0)
    {
        FILE* file = fopen(filename, "r");
        FILE* output = fopen("output.txt", "w");

        char buffer[256];
        char buffer2[256];

        int count = 0;

        // Read line by line
        while (fgets(buffer, sizeof(buffer), file))
        {
            printf("Line: %s\n", buffer);

            printf("Length of Line: %d\n", strlen(buffer));

            strcpy(buffer2, buffer);
            printf("After copying Line in Buffer 2: %s\n", buffer2);

            printf("Comparing Buffer with Buffer2: %d\n", strcmp(buffer, buffer2));

            strcat(buffer, buffer2);
            printf("After combining Lines: %s\n", buffer);

            fputs(buffer, output); // Write line to output
        }

        fclose(file);
        fclose(output);

        exit(0);
    }
    else
    {
        wait(NULL);

        printf("This is a Parent Process!\n");
    }

    return 0;
}
```

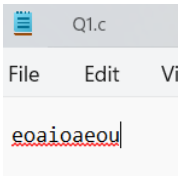
```
Line: How Are You?
Length of Line: 12
After copying Line in Buffer 2: How Are You?
Comparing Buffer with Buffer2: 0
After combining Lines: How Are You?How Are You?
This is a Parent Process!
mannanulhaq@Jarvis2024:/mnt/d/C Practice$
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <ctype.h>
#include <string.h>

int main(int argc, char* argv[])
{
    const char* filename = argv[1];

    pid_t pid = fork();

    if(pid == 0)
    {
        FILE* file = fopen(filename, "r");
        FILE* output = fopen("output.txt", "w");

        char word[100];

        int count = 0;

        // Read word by word
        while (fscanf(file, "%s", word) != EOF)
        {
            fprintf(output, "%s\n", word); // Write word by word
        }

        fclose(file);
        fclose(output);

        exit(0);
    }
    else
    {
        wait(NULL);

        printf("This is a Parent Process!\n");
    }

    return 0;
}
```




```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
```

```
#include <ctype.h>
#include <string.h>

int main(int argc, char* argv[])
{
    const char* filename = argv[1];

    pid_t pid = fork();

    if(pid == 0)
    {
        FILE* file = fopen(filename, "r");
        FILE* output = fopen("output.txt", "w");

        char ch;

        int count = 0;

        // Read character by character
        while ((ch = fgetc(file)) != EOF)
        {
            character = tolower(ch);
            // isalpha
            // isdigit
            // islower
            // isupper
            // isspace
            // toupper
            if (character == 'a' || character == 'e' || character == 'i' || character == 'o' || character =
            {
                count++;
                fputc(character, output); // Write character to output
            }
        }

        printf("Count: %d\n", count);

        fclose(file);
        fclose(output);

        exit(0);
    }
    else
    {
        wait(NULL);

        printf("This is a Parent Process!\n");
    }

    return 0;
}
```



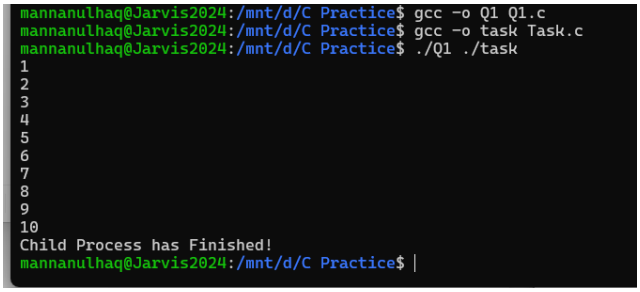
EXEC Commands:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
    pid_t pid1 = fork();

    if(pid1 == 0)
    {
        execlp("mkdir", "mkdir", "my_new_folder", NULL);

        perror("Error Occurred!");
        exit(1);
    }
    else
    {
        wait(NULL);

        pid_t pid2 = fork();

        if(pid2 == 0)
        {
            char* args[] = {"ls", "-l", NULL};
            execvp("ls", args);

            perror("Error Occurred!");
            exit(1);
        }
        else
        {
            wait(NULL);
        }
    }

    return 0;
}
```

```
// Parent Program

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char* argv[])
{
    pid_t pid = fork();

    if(pid == 0)
    {
        execlp(argv[1], argv[1], NULL);

        perror("Error Occurred!");
        exit(1);
    }
```

```
    else
    {
        wait(NULL);
        printf("Child Process has Finished!\n");
    }

    return 0;
}
```

```
// Child Program

#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```



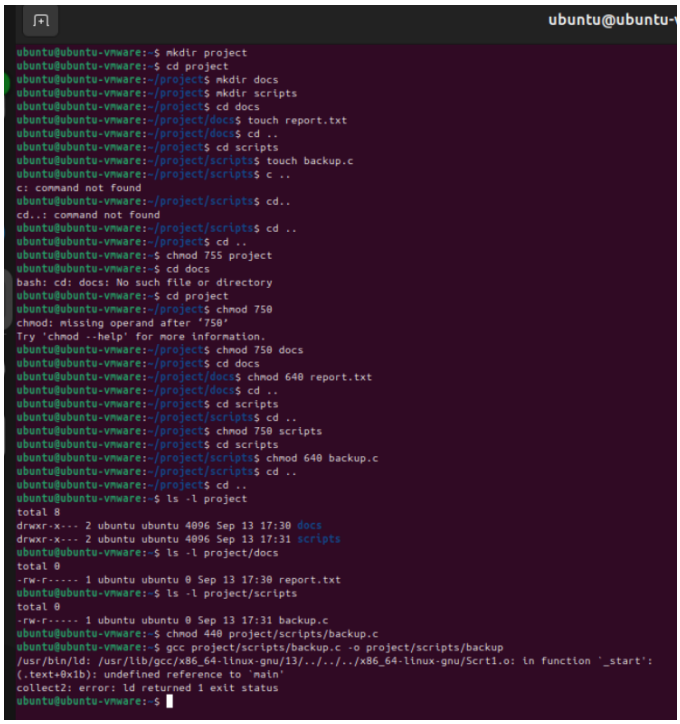
CHMOD Command:

Permission	Symbol	Octal Value
Read	`r`	`4`
Write	`w`	`2`
Execute	`x`	`1`
No Permission	`-`	`0`

Combining Permissions

The permissions can be combined by summing the values:

- **Read + Write** = `4 + 2 = 6`
- **Read + Execute** = `4 + 1 = 5`
- **Read + Write + Execute** = `4 + 2 + 1 = 7`



Un-named Pipes in Linux:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char* argv[])
{
    int N = atoi(argv[1]);

    int fd1[2], fd2[2];
    pipe(fd1);
    pipe(fd2);

    pid_t pid1 = fork();

    if(pid1 == 0)
    {
        close(fd1[0]);
        int sum = 0;

        for(int i = 1; i <= N/4; i++)
        {
            if (N % i == 0)
            {
                sum += i;
            }
        }

        write(fd1[1], &sum, sizeof(sum));
        close(fd1[1]);
        exit(0);
    }
```

```
    else
    {
        wait(NULL);

        pid_t pid2 = fork();
        if(pid2 == 0)
        {
          close(fd2[0]);
          int sum = 0;

          for(int i = N/4 + 1; i <= N/2; i++)
          {
            if (N % i == 0)
            {
                sum += i;
            }
          }

          write(fd2[1], &sum, sizeof(sum));
          close(fd2[1]);
          exit(0);
        }

        else
        {
            wait(NULL);

            close(fd1[1]);
            close(fd2[1]);

            int sum1, sum2;
            read(fd1[0], &sum1, sizeof(sum1));
            read(fd2[0], &sum2, sizeof(sum2));

            close(fd1[0]);
            close(fd2[0]);

            int total_sum = sum1 + sum2;
            if (total_sum == N)
            {
                printf("%d is a perfect number.\n", N);
            }
            else
            {
                printf("%d is not a perfect number.\n", N);
            }
        }
    }

    return 0;
}
```

Named Pipes:

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main(int argc, char* argv[])
{
    int N = atoi(argv[1]);

    int sum1, sum2;

    // Read result from fifo1
    int fd1 = open("fifo1", O_RDONLY);
    read(fd1, &sum1, sizeof(sum1));
    close(fd1);

    // Read result from fifo2
    int fd2 = open("fifo2", O_RDONLY);
    read(fd2, &sum2, sizeof(sum2));
    close(fd2);

    int total_sum = sum1 + sum2;
    if (total_sum == N)
    {
        printf("%d is a perfect number.\n", N);
    }
    else
    {
        printf("%d is not a perfect number.\n", N);
    }

    return 0;
}
```

```c
// Fifo 1

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

int main(int argc, char* argv[])
{
    int N = atoi(argv[1]);

    mkfifo("fifo1", 0644);

    int sum = 0;

    for (int i = 1; i <= N / 4; i++)
    {
        if (N % i == 0)
        {
            sum += i;
        }
    }

    int fd = open("fifo1", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    write(fd, &sum, sizeof(sum));
```

```
    close(fd);

    return 0;
}
```

```
// Fifo 2

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

int main(int argc, char* argv[])
{
    int N = atoi(argv[1]);

    mkfifo("fifo2", 0644);

    int sum = 0;

    for (int i = N / 4 + 1; i <= N / 2; i++)
    {
        if (N % i == 0)
        {
            sum += i;
        }
    }

    int fd = open("fifo2", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    write(fd, &sum, sizeof(sum));
    close(fd);

    return 0;
}
```

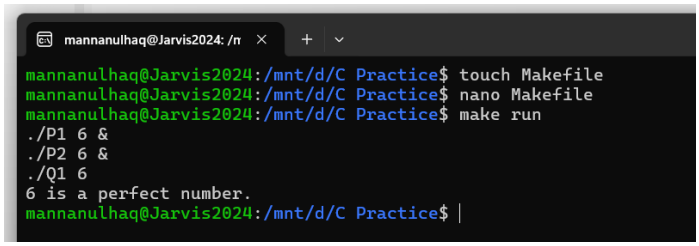
Dup Command:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>

int main()
{
    int input_fd = open("input.txt", O_RDONLY);
    int output_fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);

    dup2(input_fd, STDIN_FILENO);
    dup2(output_fd, STDOUT_FILENO);

    char c;
```

```
    int sum = 0;

    while(read(STDIN_FILENO, &c, 1) > 0)
    {
        if (isdigit(c))
        {
            sum += c - '0';
        }
    }

    printf("%d\n", sum);

    close(input_fd);
    close(output_fd);

    return 0;
}
```

Make File:

```
P1: P1.c
    gcc -o P1 P1.c -I.
P2: P2.c
    gcc -o P2 P2.c -I.
Q1: Q1.c
    gcc -o Q1 Q1.c -I.
run: P1 P2 Q1
    ./P1 6 &
    ./P2 6 &
    ./Q1 6
```



man ls | grep ls > file.txt:

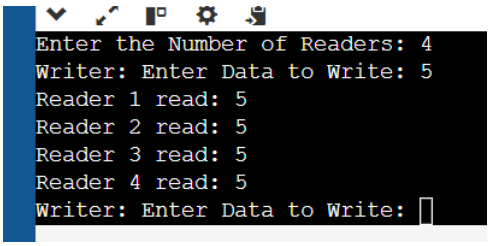
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main()
{
    int fd[2];
    pipe(fd);

    pid_t pid1 = fork();

    if(pid1 == 0)
    {
        close(fd[0]);
        dup2(fd[1], STDOUT_FILENO);
```

```
        execlp("man", "man", "ls", NULL);
    }
    else
    {
        pid_t pid2 = fork();

        if(pid2 == 0)
        {
            int file_fd = open("file.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);

            dup2(fd[0], STDIN_FILENO);
            close(fd[1]);
            close(fd[0]);

            dup2(file_fd, STDOUT_FILENO);
            close(file_fd);

            execlp("grep", "grep", "ls", NULL);
        }
    }

    return 0;
}
```

Threads and Semaphores:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

pthread_mutex_t mymutex;
sem_t writersem;

int sharedMemory = 0;
int readCount = 0;

void* readerThread(void* arg)
{
    int readerId = *(int *)arg;

    while(1)
    {
        pthread_mutex_lock(&mymutex);
        readCount++;
        if (readCount == 1)
          sem_wait(&writersem);
        pthread_mutex_unlock(&mymutex);

        printf("Reader %d read: %d\n", readerId, sharedMemory);

        pthread_mutex_lock(&mymutex);
        readCount--;
        if(readCount == 0)
          sem_post(&writersem);
        pthread_mutex_unlock(&mymutex);

        sleep(1);
    }
```

```
    pthread_exit(0);
}

void* writerThread(void* arg)
{
    while(1)
    {
        sem_wait(&writersem);

        printf("Writer: Enter Data to Write: ");
        scanf("%d", &sharedMemory);

        sem_post(&writersem);

        sleep(1);
    }

    pthread_exit(0);
}

int main()
{
    int numReaders;

    sem_init(&writersem, 0, 1);

    printf("Enter the Number of Readers: ");
    scanf("%d", &numReaders);

    pthread_t writer;
    pthread_t readers[numReaders];
    int readerIds[numReaders];

    pthread_create(&writer, NULL, writerThread, NULL);

    for (int i = 0; i < numReaders; i++)
    {
        readerIds[i] = i + 1;
        pthread_create(&readers[i], NULL, readerThread, &readerIds[i]);
    }

    pthread_join(writer, NULL);
    for (int i = 0; i < numReaders; ++i)
    {
        pthread_join(readers[i], NULL);
    }

    return 0;
}
```

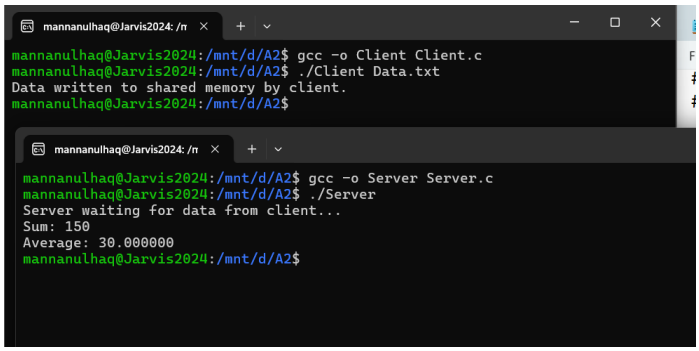


Shared Memory:

```
// Server

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define Shared_Memory_Size 1024

int main()
{
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, Shared_Memory_Size, 0666 | IPC_CREAT);
    int* data = (int* )shmat(shmid, NULL, 0);

    printf("Server waiting for data from client...\n");
    sleep(10);

    int count = data[0];
    int sum = 0;
    for(int i = 1; i <= count; i++)
    {
        sum += data[i];
    }
    float average = (float)sum / count;

    printf("Sum: %d\n", sum);
    printf("Average: %f\n", average);

    shmdt(data);
    shmctl(shmid, IPC_RMID, NULL);

    return 0;
}
```

```
// Client

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define Shared_Memory_Size 1024

int main(int argc, char* argv[])
{
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, Shared_Memory_Size, 0666);
    int* data = (int* )shmat(shmid, NULL, 0);

    FILE* file = fopen(argv[1], "r");
    int num, count = 0;
    while(fscanf(file, "%d", &num) != EOF)
    {
        data[count + 1] = num;
```

```
        count++;
    }
    data[0] = count;

    fclose(file);
    shmdt(data);

    printf("Data written to shared memory by client.\n");

    return 0;
}
```




```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <semaphore.h>
#include <time.h>
#include <fcntl.h>

#define Array_Size 10
#define Shared_Memory_Size 1024

int main()
{
    pid_t pid;
    srand(time(0));

    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, Shared_Memory_Size, 0666 | IPC_CREAT);
    int *data = (int *)shmat(shmid, NULL, 0);

    sem_t *sem_parent = sem_open("/SEM_PARENT", O_CREAT, 0666, 0);
    sem_t *sem_child = sem_open("/SEM_CHILD", O_CREAT, 0666, 0);

    pid = fork();

    if (pid > 0)
    {
        printf("Parent: Generating random numbers...\n");

        for (int i = 0; i < Array_Size; i++)
        {
            data[i] = rand() % 100;
```

```
            printf("%d ", data[i]);
        }
        printf("\n");

        sem_post(sem_child);

        sem_wait(sem_parent);

        printf("Sum = %d\n", data[Array_Size]);
        printf("Average = %f\n", *(float *)&data[Array_Size + 1]);

        shmdt(data);
        shmctl(shmid, IPC_RMID, NULL);
        sem_close(sem_parent);
        sem_close(sem_child);
        sem_unlink("/SEM_PARENT");
        sem_unlink("/SEM_CHILD");
    }
    else
    {
        sem_wait(sem_child);

        int sum = 0;
        for (int i = 0; i < Array_Size; i++)
        {
            sum += data[i];
        }
        float average = (float)sum / Array_Size;

        data[Array_Size] = sum;
        *(float *)&data[Array_Size + 1] = average;

        sem_post(sem_parent);

        shmdt(data);
        sem_close(sem_parent);
        sem_close(sem_child);
    }

    return 0;
}
```

File Mapping:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <ctype.h>

size_t map_size = 100;

void* replace_integers(void* arg)
{
    char* map = (char* )arg;
    for (int i = 0; i < map_size / 2; i++)
    {
```

```c
        if (isdigit(map[i]))
        {
            map[i] = ' ';
        }
    }

    pthread_exit(0);
}

int main(int argc, char* argv[])
{
    int fd = open(argv[1], O_RDWR);
    void* file_memory = mmap(NULL, map_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

    printf("Original file contents:\n%s\n", (char *)file_memory);

    pthread_t thread1, thread2;

    pthread_create(&thread1, NULL, replace_integers, file_memory);
    pthread_create(&thread2, NULL, replace_integers, (char *)file_memory + map_size / 2);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("Modified file contents:\n%s\n", (char *)file_memory);

    munmap(file_memory, map_size);
    close(fd);

    return 0;
}
```

```
mannanulhaq@Jarvis2024:/mnt/d/C Practice$ gcc -o P1 P1.c
mannanulhaq@Jarvis2024:/mnt/d/C Practice$ ./P1 Input.txt
Original file contents:
vigU6OTgN7DMifG7zmQWp04ZEyGmRifq1uFsS9RzZWcCQL7jBMNkUQVEAIsKZia40M3TqJeGEMEkkSagfUc7mU3PbQ1zsiJm23H
Modified file contents:
v gU OTgN DMifG zmQWp  ZEyGmRifq uFsS RzZWcCQL jBMNkUQVEAIsKZia  M TqJeGEMEkkSagfUc mU PbQ zsiJm  H
mannanulhaq@Jarvis2024:/mnt/d/C Practice$
```