Child Process Uygulamaları Ödevi

1. Child Process Oluşturma ve Yönetimi

Amaç: Parent process bir child process oluşturacak. Child process bir dosyanın içeriğini ekrana yazdıracak. Parent process, child process'in tamamlanmasını bekleyecek ve çıkış durumunu kontrol edecek.

Kod Çözümü:

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

int main() {
    pid_t pid = fork(); // Child process oluştur
    if (pid == -1) {
        perror("Fork failed");
        return 1;
    }
    if (pid == 0) {
        // Child process
        execlp("cat", "cat", "file.txt", NULL); // file.txt içeriğini
ekrana yaz
        perror("execlp failed"); // execlp başarısız olursa
        exit(1);
    } else {
        // Parent process
        int status;
        wait(&status); // Child process'in bitmesini bekle
        if (WIFEXITED(status)) {
            printf("Child process exit status: %d\n",
WEXITSTATUS(status));
        } else {
            printf("Child process did not exit normally.\n");
        }
```

```
    }
    return 0;
}
```

---

2. Child Process’lerde Hata Yönetimi

Amaç: Child process bir dosyayı açmaya çalışacak. Dosya bulunamazsa abort() ile çıkacak, bulunursa exit(0) ile düzgün çıkacak. Parent process durumu analiz edecek.

Kod Çözümü:

```
#include <stdio.h>

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

int main() {
    pid_t pid = fork(); // Child process oluştur
    if (pid == -1) {
        perror("Fork failed");
        return 1;
    }
    if (pid == 0) {
        // Child process
        FILE *file = fopen("file.txt", "r");
        if (file == NULL) {
            perror("File not found");
            abort(); // Zorla çıkış
        }
        printf("File opened successfully.\n");
        fclose(file);
        exit(0); // Başarıyla çık
    } else {
        // Parent process
        int status;
```

```
        wait(&status);
        if (WIFSIGNALED(status)) {
            printf("Child process terminated by signal: %d\n",
WTERMSIG(status));
        } else if (WIFEXITED(status)) {
            printf("Child process exit status: %d\n",
WEXITSTATUS(status));
        } else {
            printf("Child process did not terminate normally.\n");
        }
    }
    return 0;
}
```

---

3. Paralel Child Process’lerle Çoklu Görev Yönetimi

Amaç: Parent process aynı anda 3 child process oluşturacak. İlk child bir dosya oluşturacak, ikinci child veri ekleyecek, üçüncü child dosyayı okuyup içeriği yazacak. Parent process tüm child process'leri bekleyecek.

Kod Çözümü:

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

void create_file() {
    FILE *file = fopen("file.txt", "w");
    if (file == NULL) {
        perror("File creation failed");
        exit(1);
    }
    fprintf(file, "Initial content\n");
    fclose(file);
    exit(0);
}
```

```
void write_file() {
    FILE *file = fopen("file.txt", "a");
    if (file == NULL) {
        perror("File open failed");
        exit(1);
    }
    fprintf(file, "Appended content\n");
    fclose(file);
    exit(0);
}

void read_file() {
    char buffer[256];
    FILE *file = fopen("file.txt", "r");
    if (file == NULL) {
        perror("File open failed");
        exit(1);
    }
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        printf("%s", buffer);
    }
    fclose(file);
    exit(0);
}

int main() {
    pid_t pid1, pid2, pid3;

    // First child
    pid1 = fork();
    if (pid1 == 0) create_file();

    // Second child
    pid2 = fork();
    if (pid2 == 0) write_file();

    // Third child
```

```
    pid3 = fork();
    if (pid3 == 0) read_file();

    // Parent process
    wait(NULL); // Wait for first child
    wait(NULL); // Wait for second child
    wait(NULL); // Wait for third child

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