**IPC: Interrupts and Signals**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 2e**

**Title-** Write a program to send signal by five different signal sending system calls and identify the difference in working with example.

**Objectives-**

1. To learn about IPC through signal.
2. To know the process management of Unix/Linux OS
3. Use of system call to write effective application programs

**Theory-**

This program demonstrates sending signals using five different signal-sending system calls in Linux. The program highlights the differences in how these system calls work, with examples and explanations for each.

Signal handling is a fundamental aspect of process communication and control in UNIX-based systems. This document includes pseudocode and a full C implementation.

The following signal-sending system calls are covered:  
1. `kill()`: Sends a signal to a specific process or group of processes.  
2. `raise()`: Sends a signal to the process that calls it.  
3. `sigqueue()`: Sends a real-time signal with additional data to a specific process.  
4. `pthread\_kill()`: Sends a signal to a specific thread within a process.  
5. `alarm()`: Sends a SIGALRM signal after a specified amount of time.

**Program-**

#include <stdio.h>  
#include <stdlib.h>  
#include <signal.h>  
#include <unistd.h>  
#include <pthread.h>  
  
// Signal handler  
void signal\_handler(int signum) {  
 printf("Received signal: %d\n", signum);  
}  
  
// Thread function  
void \*thread\_function(void \*arg) {  
 printf("Thread sending SIGUSR2 to itself using pthread\_kill()...\n");  
 pthread\_kill(pthread\_self(), SIGUSR2);  
 return NULL;  
}  
  
int main() {  
 // Set up signal handlers  
 signal(SIGUSR1, signal\_handler);  
 signal(SIGUSR2, signal\_handler);  
 signal(SIGALRM, signal\_handler);  
  
 pid\_t pid = fork();  
  
 if (pid == 0) {  
 // Child process  
 printf("Child: Raising SIGUSR1 using raise()...\n");  
 raise(SIGUSR1);  
 } else {  
 // Parent process  
 printf("Parent: Sending SIGUSR1 to child using kill()...\n");  
 kill(pid, SIGUSR1);  
  
 printf("Parent: Sending real-time signal using sigqueue()...\n");  
 sigqueue(pid, SIGUSR2, (union sigval){.sival\_int = 42});  
  
 printf("Parent: Setting alarm for 2 seconds using alarm()...\n");  
 alarm(2);  
  
 pthread\_t thread;  
 printf("Parent: Creating a thread to demonstrate pthread\_kill()...\n");  
 pthread\_create(&thread, NULL, thread\_function, NULL);  
 pthread\_join(thread, NULL);  
 }  
  
 // Sleep to allow signals to be processed  
 sleep(3);  
 return 0;  
}

**Conclusion-**

This program demonstrates the use of five different signal-sending system calls in Linux. Each call has unique features and use cases, providing flexibility for signal handling in various scenarios. Understanding these differences is crucial for effective process and thread communication.