#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <time.h>

// Global variable to hold the child process ID

pid\_t child\_pid = -1;

// Signal handler for child process death

void child\_signal\_handler(int sig) {

int status;

// Wait for the child process to exit

if (child\_pid != -1) {

waitpid(child\_pid, &status, 0);

printf("Child process (PID: %d) has terminated.\n", child\_pid);

}

}

// Signal handler for alarm (timeout)

void alarm\_signal\_handler(int sig) {

if (child\_pid != -1) {

printf("Child process did not finish in time. Killing it...\n");

kill(child\_pid, SIGKILL); // Kill the child process

}

}

int main() {

struct sigaction sa;

// Set up the signal handler for child termination

sa.sa\_handler = child\_signal\_handler;

sa.sa\_flags = 0;

sigemptyset(&sa.sa\_mask);

sigaction(SIGCHLD, &sa, NULL);

// Set up the signal handler for alarm (timeout)

sa.sa\_handler = alarm\_signal\_handler;

sigaction(SIGALRM, &sa, NULL);

// Fork a child process

child\_pid = fork();

if (child\_pid == -1) {

perror("Fork failed");

exit(1);

}

if (child\_pid == 0) {

// Child process

printf("Child process (PID: %d) started...\n", getpid());

// Run a user-defined program or command, for example 'sleep 10'

// Replace this with the desired command to run in the child

execlp("sleep", "sleep", "10", (char \*)NULL);

// If execlp fails

perror("execlp failed");

exit(1);

} else {

// Parent process

printf("Parent process (PID: %d) will wait for child.\n", getpid());

// Set an alarm to kill the child if it takes more than 5 seconds

alarm(5);

// Wait for the child to complete

pause(); // Wait for signals (SIGCHLD or SIGALRM)

printf("Parent process is terminating.\n");

}

return 0;

}