Coding

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

volatile sig_atomic_t interrupted = 0;

void sigint_handler(int signum)
{
interrupted = 1;
}

int main()
{
int i, pid, fd[2];
char message[20];

// Set up signal handler for interrupt signal
struct sigaction sa;
sa.sa_handler = sigint_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0;
sigaction(SIGINT, &sa, NULL);

if (pipe(fd) == -1) {
printf("Error: Failed to create pipe.\n");
exit(1);
}

printf("Enter a message to send to the child processes: ");
fgets(message, 20, stdin);

for (i = 0; i < 5; i++) {
pid = fork();
if (pid == -1) {
printf("Error: Failed to create child process.\n");
exit(1);
```

```
}
else if (pid == 0) {
// Child process code
close(fd[1]); // close unused write end of the pipe
read(fd[0], message, 20);
printf("Child process %d with PID %d received message: %s\n", i+1, getpid(), message);
exit(0);
}
else {
// Parent process code
printf("Parent process with PID %d created child process %d with PID %d\n", getpid(), i+1, pid);
close(fd[0]); // close unused read end of the pipe
write(fd[1], message, 20);
}
}

// Wait for all child processes to finish or interrupt signal to be received
while (!interrupted && i > 0) {
wait(NULL);
i--;
}

// Check if program was interrupted
if (interrupted) {
printf("Program interrupted. Exiting.\n");
}
else {
printf("All child processes finished. Parent process exiting.\n");
}

return 0;
}
```