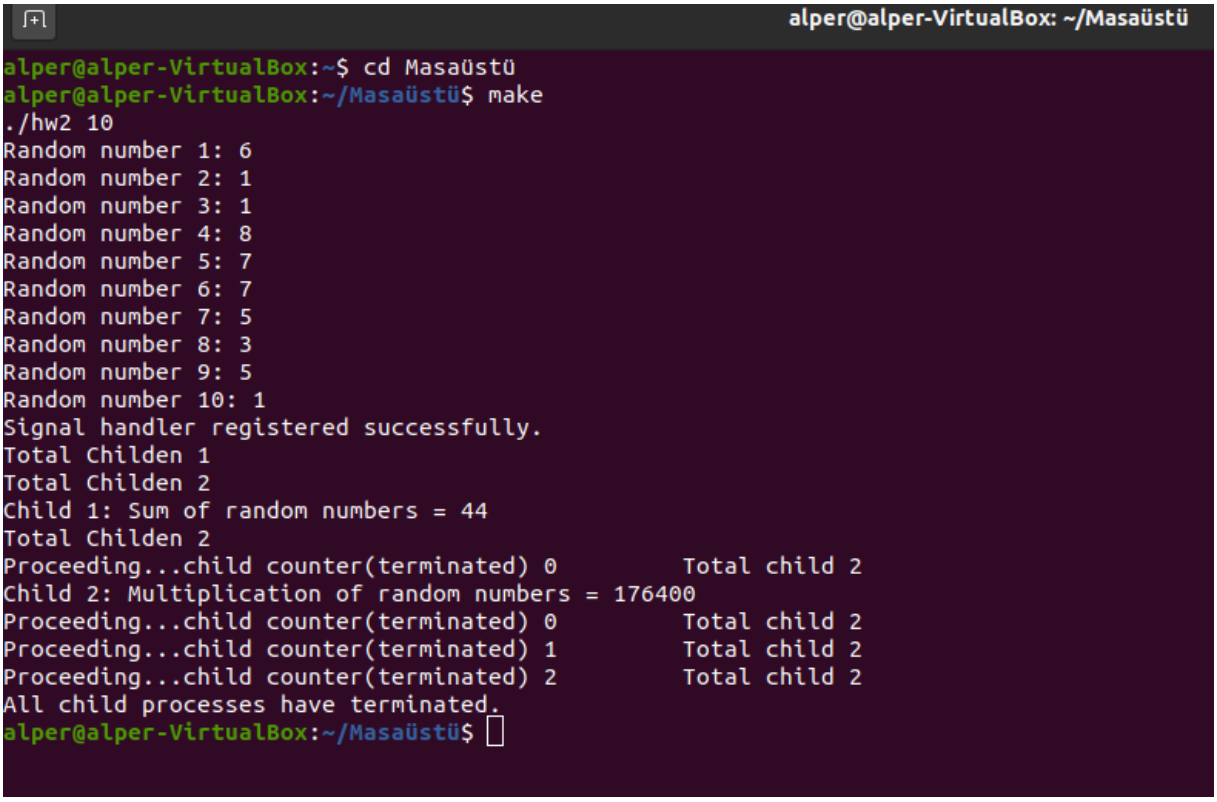
Gebze Technical University
Department of Computer Engineering
CSE344 - Spring 2024
Homework 2

AHMET ALPER UZUNTEPE
1901042669



The objective of this assignment is to demonstrate the usage of FIFO (First-In-First-Out) communication between parent and child processes in C programming. Additionally, signal handling for the termination of child processes using the SIGCHLD signal is implemented.

Overview

- The program takes an integer argument and representing the number of random numbers to generate.
- Two FIFOs (FIFO1 and FIFO2) are created using mkfifo() function for communication between parent and child processes.
- Random numbers are generated and displayed.
- Signal handling for SIGCHLD signal is set up using sigaction() to handle termination of child processes.
- Two child processes are created using fork(). Each child performs different tasks based on its index:
- Child 1: Reads random numbers from FIFO1 and calculates their sum.
- Child 2: Reads a command ("multiply") from FIFO2 sent by the parent. If the command is "multiply", it reads random numbers from FIFO1 and calculates their multiplication.
- Parent process sends random numbers to FIFO1 and the "multiply" command to FIFO2 for the second child.
- The parent process waits for child processes to terminate using wait(). Upon termination of a child process, the SIGCHLD signal is caught and handled. The counter for terminated children is incremented, and the loop continues until all child processes have terminated.
- Finally, the FIFOs are unlinked using unlink().

HW2.c -

File Edit Selection View Go Run Terminal Help

HW2.c

home > alper > Masaüstü > HW2.c > main(int, char * [])

```c
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/stat.h>
#include <time.h>
#include <errno.h>

#define FIFO1 "fifo1"
#define FIFO2 "fifo2"
#define MAX_CHILDREN 2

int main(int argc, char *argv[])
{
  if (argc != 2)
  {
    printf("Usage: %s <integer>\n", argv[0]);
    exit(1);
  }
  int child_counter = 0;  // counter for terminated children
  int total_children = 0; // counter for total child processes created
  int num = atoi(argv[1]);
  int random_numbers[num];

  // create fifo
  mkfifo(FIFO1, 0666);
  mkfifo(FIFO2, 0666);

  // random numbers
  srand(time(NULL));
  for (int i = 0; i < num; i++)
  {
    random_numbers[i] = rand() % 10;
    printf("Random number %d: %d\n", i + 1, random_numbers[i]);
  }

  // set up sigaction for SIGCHLD
  struct sigaction sa;
  sigemptyset(&sa.sa_mask);
  sa.sa_flags = SA_RESTART; // restart interrupted system calls
  if (sigaction(SIGCHLD, &sa, NULL) == -1)
  {
    perror("sigaction");
    exit(1);
  }
  printf("Signal handler registered successfully.\n");
```

⊗ 0 ⚠ 0

Signal Handling:

The SIGCHLD signal is handled using the sigaction() function with SA_RESTART flag. This ensures that system calls interrupted by this signal are restarted.

Upon receiving the SIGCHLD signal, the parent process increments the counter for terminated children and continues waiting for further terminations until all child processes have terminated.

- Sets up signal handling for SIGCHLD, ensuring interrupted system calls are restarted.

```c
// create fifo
mkfifo(FIFO1, 0666);
mkfifo(FIFO2, 0666);

// random numbers
srand(time(NULL));
for (int i = 0; i < num; i++)
{
  random_numbers[i] = rand() % 10;
  printf("Random number %d: %d\n", i + 1, random_numbers[i]);
}

// set up sigaction for SIGCHLD
struct sigaction sa;
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART; // restart interrupted system calls
if (sigaction(SIGCHLD, &sa, NULL) == -1)
{
  perror("sigaction");
  exit(1);
}
printf("Signal handler registered successfully.\n");
```

```
Signal handler registered successfully.
Total Childen 1
Total Childen 2
Child 1: Sum of random numbers = 72
Total Childen 2
Proceeding...child counter(terminated) 0         Total child 2
Proceeding...child counter(terminated) 0         Total child 2
Child 2: Multiplication of random numbers = 300056400
Proceeding...child counter(terminated) 1         Total child 2
Proceeding...child counter(terminated) 2         Total child 2
All child processes have terminated.
alper@alper-VirtualBox:~/Masaüstü$
```

- Enters an infinite loop to monitor and terminate the child processes with using sleep(2)(for delay).This loop count the child which one is terminated and compare it with total number of childs.If both of them terminated break the loop and print the status of childs.

```c
  while (1)
  {
    printf("Proceeding...child counter(terminated) %d \t Total child %d \n", child_counter, total_children);
    sleep(2); // delay
    if (child_counter < total_children)
    {
      // wait child process to terminate
      pid_t terminated_pid = wait(NULL);

      if (terminated_pid > 0)
      {
        // printf("child process pid %d terminated with child count %d\n", terminated_pid, child_counter);
        child_counter++;
      }
      else if (terminated_pid == 0)
      {
        printf("No child processes have terminated.\n");
      }
      else
      {
        if (terminated_pid == -1 && errno == ECHILD)
        {
          break;
        }
        else
        {
          printf("Error occurred during waitpid()\n");
          perror("waitpid");
          exit(1);
        }
      }
    }
    else
    {
      break;
    }
  }
  // printf("childcounter %d", child_counter);
  printf("All child processes have terminated.\n");

  // remove unlink fifos
  unlink(FIFO1);
  unlink(FIFO2);

  return 0;
}
```

Child Process

- The code begins by iterating through a loop to create child processes. The number of child processes to create is defined by MAX_CHILDREN.
- For each iteration, the total count of child processes (total_children) is incremented.
- Inside the loop, the fork() system call is used to create a child process.
- If fork() is successful, the child process executes the code inside the else if (pid == 0) block, while the parent process continues to execute the code in the else block.
- Each child process sleeps for 10 seconds to simulate a delay, as specified in the assignment requirements.
- If the child process is the first one, it reads random numbers from FIFO1, calculates their sum, and prints the result.
- If the child process is the second one ,it reads a command from FIFO2. If the command is "multiply," it reads random numbers from FIFO1, calculates their multiplication, and prints the result. Otherwise, it prints an error message and exits.

HW2.c

home > alper > Masaüstü > HW2.c > main(int, char * [])

```c

    // create child process
    for (int i = 0; i < MAX_CHILDREN; i++)
    {
      total_children++; // increment total child process counter
      printf("Total Childen %d\n", total_children);
      pid_t pid = fork();
      if (pid == -1)
      {
        perror("fork");
        exit(1);
      }
      else if (pid == 0)
      {           // child process
        sleep(10); // sleep for 10 seconds (in the pdf given 10 secs)
        if (i == 0)
        { // 1. child
          int fd;
          int sum = 0;

          // open fif 1 for reading
          fd = open(FIFO1, O_RDONLY);
          if (fd == -1)
          {
            perror("open (FIFO1)");
            exit(1);
          }

          // read random numbers from fifo 1
          for (int j = 0; j < num; j++)
          {
            int n;
            if (read(fd, &n, sizeof(int)) == -1)
            {
              perror("read (FIFO1)");
              exit(1);
            }
            sum += n;
          }
          close(fd); // close the read olny fifo for 1.child
          printf("Child 1: Sum of random numbers = %d\n", sum);
        }
        else
        { // 2. child
          int fd;
          char command[10];
          int result = 1;
```

Parent Process

- The parent process handles communication with the child processes.
- If it's the first iteration of the loop, it writes random numbers to FIFO1.
- For subsequent iterations ,it writes random numbers to FIFO1 and sends the "multiply" command to FIFO2.
- Error handling ensures that file operations succeed, and appropriate error messages are printed if there are failures.

```
else
{ // parent process
  if (i == 0)
  { // for the first child
    // write random numbers to fifo 1
    int fd = open(FIFO1, O_WRONLY);
    if (fd == -1)
    {
      perror("open (FIFO1)");
      exit(1);
    }
    for (int i = 0; i < num; i++)
    {
      if (write(fd, &random_numbers[i], sizeof(int)) == -1)
      {
        perror("write (FIFO1)");
        exit(1);
      }
    }
    close(fd);
  }
  else
  { // for the second child
    // write random numbers to fifo 1

    int fd = open(FIFO1, O_WRONLY);
    if (fd == -1)
    {
      perror("open (FIFO1)");
      exit(1);
    }
    for (int i = 0; i < num; i++)
    {
      if (write(fd, &random_numbers[i], sizeof(int)) == -1)
      {
        perror("write (FIFO1)");
        exit(1);
      }
    }
    close(fd);

    // write "multiply" command to fifo 2
    fd = open(FIFO2, O_WRONLY);
    if (fd == -1)
    {
      perror("open (FIFO2)");
      exit(1);
```

Interprocess Communication

- FIFOs are used for communication between the parent and child processes.
- FIFO1 is used to pass random numbers from the parent to the child processes.
- FIFO2 is used to send commands (specifically "multiply") from the parent to the second child process.
- The parent process cleans up by closing file descriptors and removing/unlinking FIFOs after all child processes have terminated.(unlink)

File Edit Selection View Go Run Terminal Help

HW2.c

home > alper > Masaüstü > HW2.c > main(int, char * [])

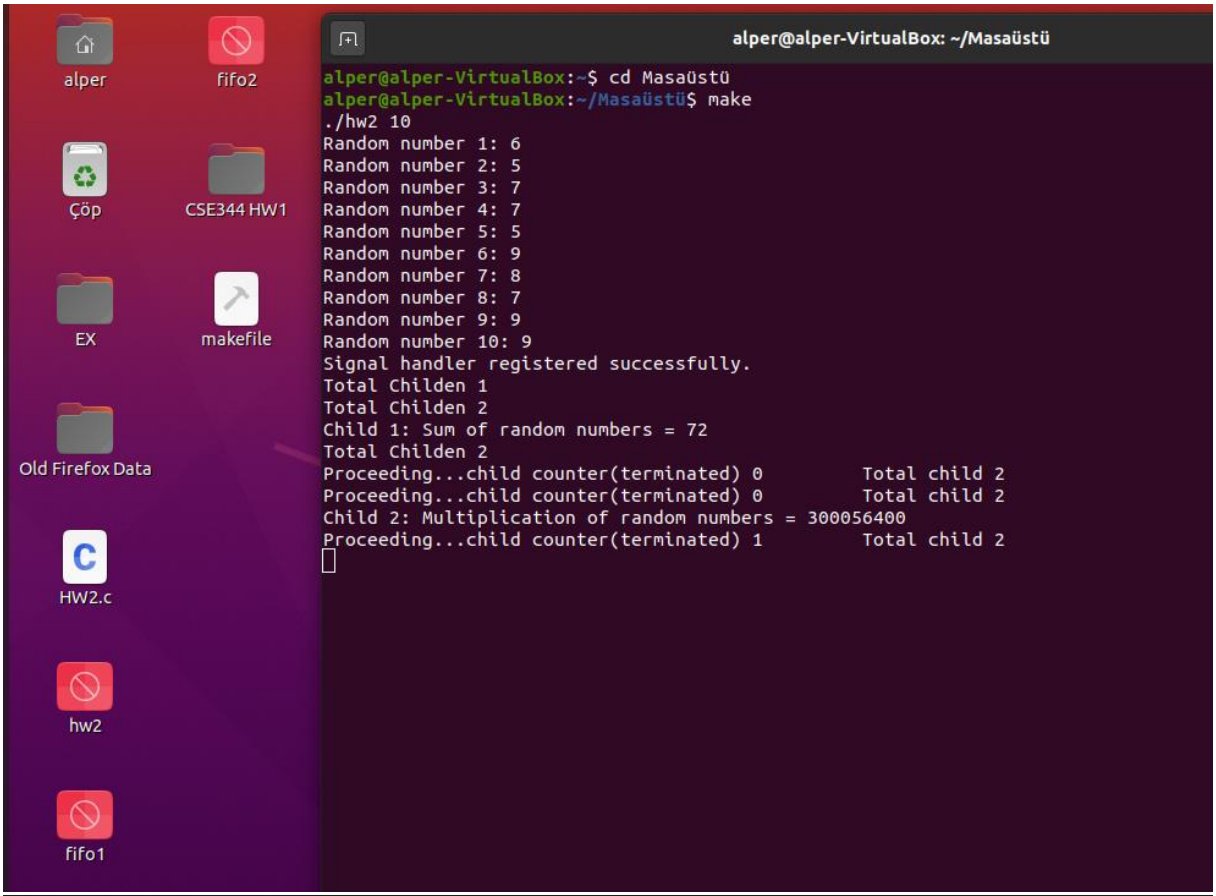
```c
{
    if (argc != 2)
    {
        printf("Usage: %s <integer>\n", argv[0]);
        exit(1);
    }
    int child_counter = 0;  // conunter for terminated children
    int total_children = 0; // conunter for total child processes created
    int num = atoi(argv[1]);
    int random_numbers[num];

    // create fifo
    mkfifo(FIFO1, 0666);
    mkfifo(FIFO2, 0666);

    // random numbers
    srand(time(NULL));
    for (int i = 0; i < num; i++)
    {
        random_numbers[i] = rand() % 10;
        printf("Random number %d: %d\n", i + 1, random_numbers[i]);
    }

    // set up sigaction for SIGCHLD
    struct sigaction sa;
```

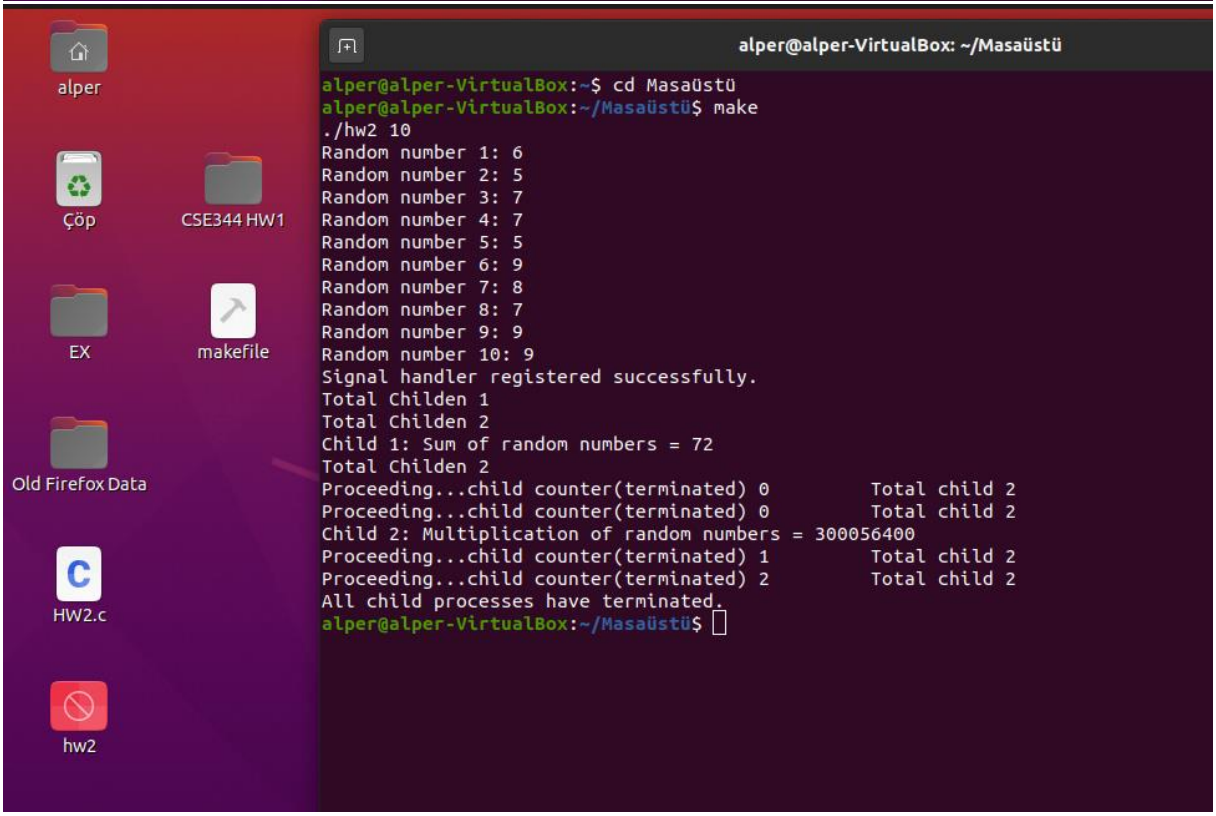
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Child 2: Multiplication of random numbers = 0
Proceeding...child counter(terminated) 0         Total child 2
Proceeding...child counter(terminated) 1         Total child 2
Proceeding...child counter(terminated) 2         Total child 2
All child processes have terminated.
alper@alper-VirtualBox:~/Masaüstü$ ./hw2 6
Random number 1: 3
Random number 2: 4
Random number 3: 3
Random number 4: 1
Random number 5: 3
Random number 6: 6
Signal handler registered successfully.
Total Childen 1
Total Childen 2
Child 1: Sum of random numbers = 20
Total Childen 2
Proceeding...child counter(terminated) 0         Total child 2
Child 2: Multiplication of random numbers = 648
Proceeding...child counter(terminated) 0         Total child 2
Proceeding...child counter(terminated) 1         Total child 2
Proceeding...child counter(terminated) 2         Total child 2
All child processes have terminated.
alper@alper-VirtualBox:~/Masaüstü$
```

⊗ 0 ⚠ 0

OUTPUT


alper
fifo2
Çöp
CSE344 HW1
EX
makefile
Old Firefox Data
HW2.c
hw2
fifo1
alper@alper-VirtualBox: ~/Masaüstü
alper@alper-VirtualBox:~$ cd Masaüstü
alper@alper-VirtualBox:~/Masaüstü$ make
./hw2 10
Random number 1: 6
Random number 2: 5
Random number 3: 7
Random number 4: 7
Random number 5: 5
Random number 6: 9
Random number 7: 8
Random number 8: 7
Random number 9: 9
Random number 10: 9
Signal handler registered successfully.
Total Childen 1
Total Childen 2
Child 1: Sum of random numbers = 72
Total Childen 2
Proceeding...child counter(terminated) 0 Total child 2
Proceeding...child counter(terminated) 0 Total child 2
Child 2: Multiplication of random numbers = 300056400
Proceeding...child counter(terminated) 1 Total child 2


alper
Çöp
CSE344 HW1
EX
makefile
Old Firefox Data
HW2.c
hw2
alper@alper-VirtualBox: ~/Masaüstü
alper@alper-VirtualBox:~$ cd Masaüstü
alper@alper-VirtualBox:~/Masaüstü$ make
./hw2 10
Random number 1: 6
Random number 2: 5
Random number 3: 7
Random number 4: 7
Random number 5: 5
Random number 6: 9
Random number 7: 8
Random number 8: 7
Random number 9: 9
Random number 10: 9
Signal handler registered successfully.
Total Childen 1
Total Childen 2
Child 1: Sum of random numbers = 72
Total Childen 2
Proceeding...child counter(terminated) 0 Total child 2
Proceeding...child counter(terminated) 0 Total child 2
Child 2: Multiplication of random numbers = 300056400
Proceeding...child counter(terminated) 1 Total child 2
Proceeding...child counter(terminated) 2 Total child 2
All child processes have terminated.
alper@alper-VirtualBox:~/Masaüstü$

CHECK MEMORY LEAK

```
alper@alper-VirtualBox: ~/Masaüstü
Random number 6: 5
Random number 7: 9
Random number 8: 0
Random number 9: 1
Random number 10: 2
==2488== Syscall param rt_sigaction(act->sa_handler) points to uninitialised byte(s)
==2488==    at 0x48A7166: __libc_sigaction (sigaction.c:58)
==2488==    by 0x10963D: main (in /home/alper/Masaüstü/hw2)
==2488==  Address 0x1ffefffc50 is on thread 1's stack
==2488==  in frame #0, created by __libc_sigaction (sigaction.c:43)
==2488==
Signal handler registered successfully.
Total Childen 1
Total Childen 2
Child 1: Sum of random numbers = 35
Total Childen 2
Proceeding...child counter(terminated) 0         Total child 2
Proceeding...child counter(terminated) 0         Total child 2
Child 2: Multiplication of random numbers = 0
==2494==
==2494== HEAP SUMMARY:
==2494==     in use at exit: 0 bytes in 0 blocks
==2494==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==2494==
==2494== All heap blocks were freed -- no leaks are possible
==2494==
==2494== Use --track-origins=yes to see where uninitialised values come from
==2494== For lists of detected and suppressed errors, rerun with: -s
==2494== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Proceeding...child counter(terminated) 1         Total child 2
Proceeding...child counter(terminated) 2         Total child 2
All child processes have terminated.
==2492==
==2492== HEAP SUMMARY:
==2492==     in use at exit: 0 bytes in 0 blocks
==2492==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==2492==
==2492== All heap blocks were freed -- no leaks are possible
==2492==
==2492== Use --track-origins=yes to see where uninitialised values come from
==2492== For lists of detected and suppressed errors, rerun with: -s
==2492== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Proceeding...child counter(terminated) 1         Total child 2
Proceeding...child counter(terminated) 2         Total child 2
All child processes have terminated.
==2488==
==2488== HEAP SUMMARY:
==2488==     in use at exit: 0 bytes in 0 blocks
==2488==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==2488==
==2488== All heap blocks were freed -- no leaks are possible
==2488==
==2488== Use --track-origins=yes to see where uninitialised values come from
==2488== For lists of detected and suppressed errors, rerun with: -s
==2488== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
alper@alper-VirtualBox:~/Masaüstü$
```