# CSE-344 SYSTEM PROGRAMMING

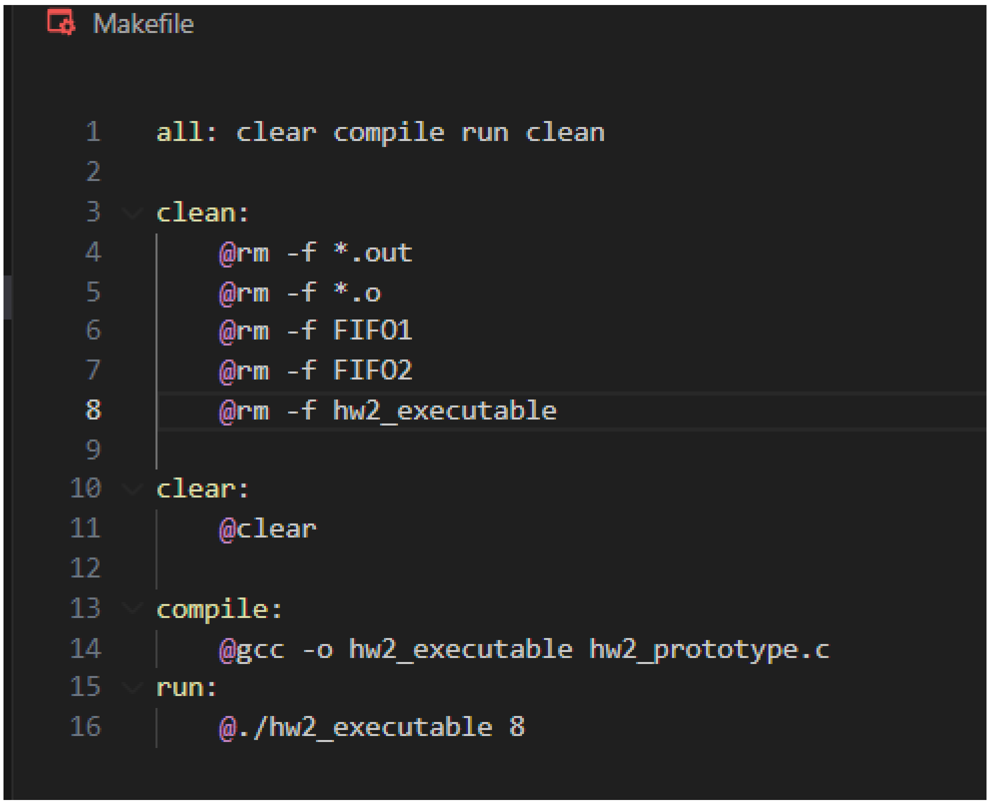
SPRING 2024

HOMEWORK 2

REPORT

ONUR ATASEVER

210104004087



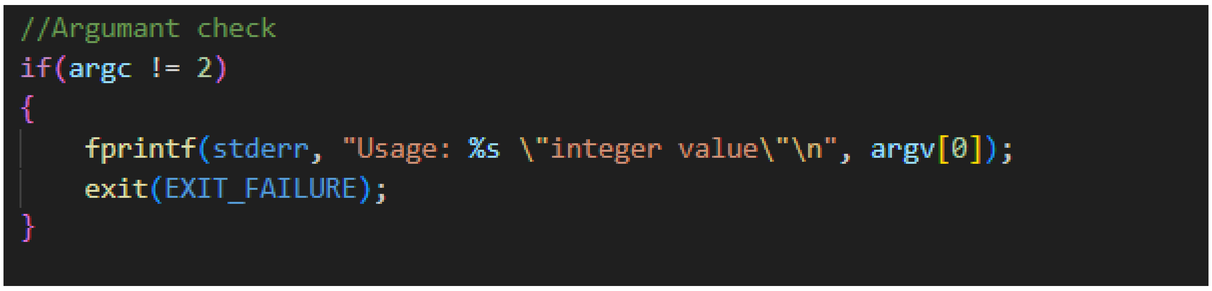
Makefile

```
all: clear compile run clean

clean:
	@rm -f *.out
	@rm -f *.o
	@rm -f FIFO1
	@rm -f FIFO2
	@rm -f hw2_executable

clear:
	@clear

compile:
	@gcc -o hw2_executable hw2_prototype.c
run:
	@./hw2_executable 8
```

To run this program you can use “make” command. To specify number of randoms you can change the integer argument in the run section. Created FIFOs are unlinked end of the program but in case of any unexpected termination, FIFOs are removed in makefile.

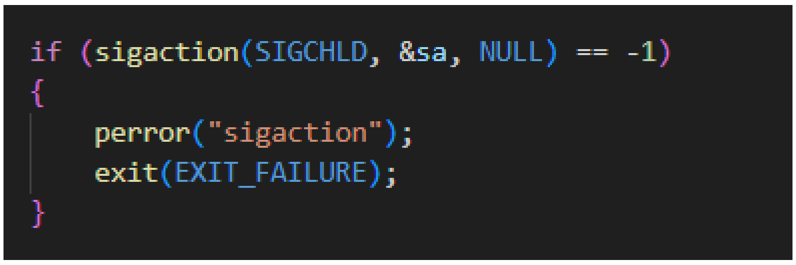
## Error Checks:

If the number of argument is different than 2 it exits.



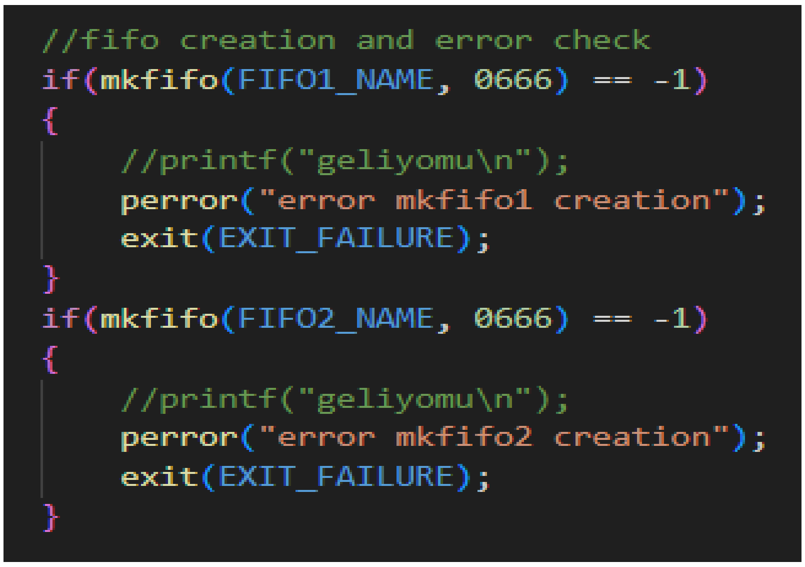
```c
//Argumant check
if(argc != 2)
{
    fprintf(stderr, "Usage: %s \"integer value\"\n", argv[0]);
    exit(EXIT_FAILURE);
}
```

if the return value of **sigaction** is -1, it means that an error occurred.



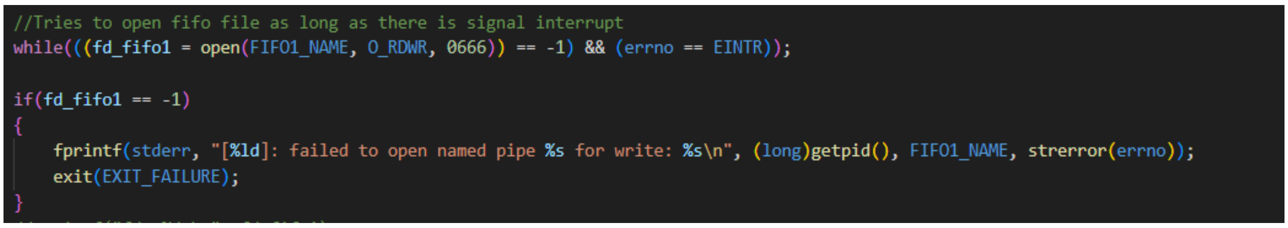
```c
if (sigaction(SIGCHLD, &sa, NULL) == -1)
{
    perror("sigaction");
    exit(EXIT_FAILURE);
}
```

If any error occurs while creating FIFOs, it exits with error message.



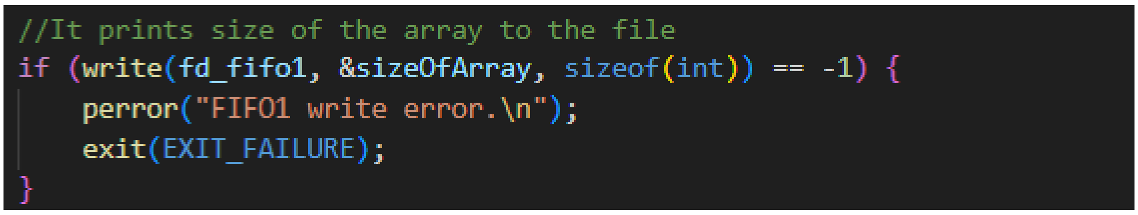
```c
//fifo creation and error check
if(mkfifo(FIFO1_NAME, 0666) == -1)
{
    //printf("geliyomu\n");
    perror("error mkfifo1 creation");
    exit(EXIT_FAILURE);
}
if(mkfifo(FIFO2_NAME, 0666) == -1)
{
    //printf("geliyomu\n");
    perror("error mkfifo2 creation");
    exit(EXIT_FAILURE);
}
```

If there is any error other then EINTR it prints and exits. For other file openin system calls, there are same controls.



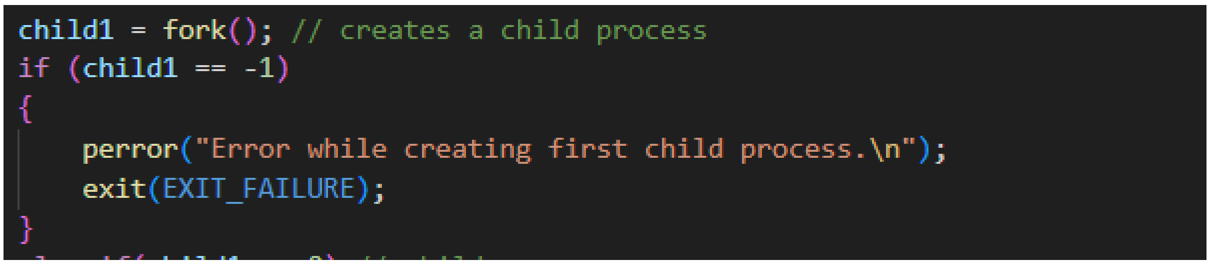
```c
//Tries to open fifo file as long as there is signal interrupt
while(((fd_fifo1 = open(FIFO1_NAME, O_RDWR, 0666)) == -1) && (errno == EINTR));

if(fd_fifo1 == -1)
{
    fprintf(stderr, "[%ld]: failed to open named pipe %s for write: %s\n", (long)getpid(), FIFO1_NAME, strerror(errno));
    exit(EXIT_FAILURE);
}
```

If there is any error while writing or reading, it exits with error message. For other write and read system calls, there are same controls.



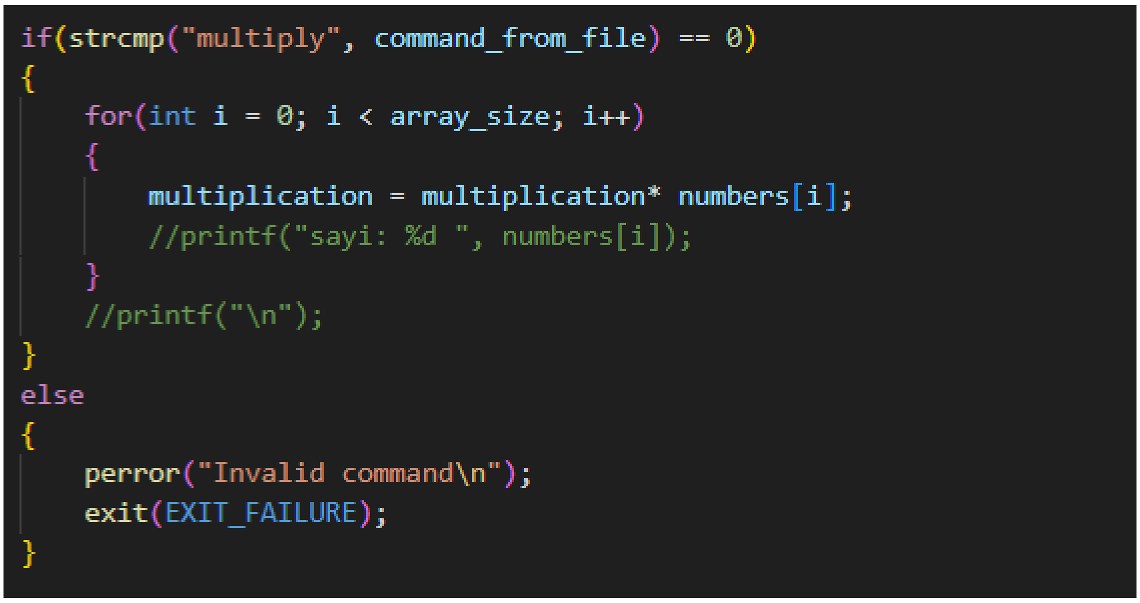
```c
//It prints size of the array to the file
if (write(fd_fifo1, &sizeOfArray, sizeof(int)) == -1) {
    perror("FIFO1 write error.\n");
    exit(EXIT_FAILURE);
}
```

If any error occurs while creatin new process with fork system call, it exits with error message. For other fork operation, there is same control.



```c
child1 = fork(); // creates a child process
if (child1 == -1)
{
    perror("Error while creating first child process.\n");
    exit(EXIT_FAILURE);
}
```

It checks if the command read from file is “multiply”. If it is not, it exits with error message.



```c
if(strcmp("multiply", command_from_file) == 0)
{
    for(int i = 0; i < array_size; i++)
    {
        multiplication = multiplication* numbers[i];
        //printf("sayi: %d ", numbers[i]);
    }
    //printf("\n");
}
else
{
    perror("Invalid command\n");
    exit(EXIT_FAILURE);
}
```

## Program Flow:

1-) Parent process sets signal handler for SIGCHLD

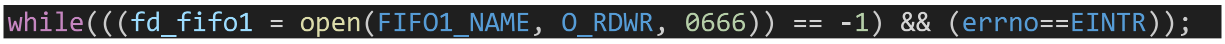
2-) According to second argument, memory is allocated for random number by using malloc.

3-) Random array is filled according to size of array:

randomNumbersArray[i] = rand()%sizeOfArray;

4-) FIFOs are created.

5-) In the parent process, FIFO1 file is opened and informations about array is written to the file. As long as there is signal interrupt, it tries to open the file with system calls.



```c
while(((fd_fifo1 = open(FIFO1_NAME, O_RDWR, 0666)) == -1) && (errno==EINTR));
```

6-) In the parent process, FIFO2 file is opened and informations about command and array is written to the file. As long as there is signal interrupt, it tries to open the file with system calls.

7-) Parent process creates a new child by using fork syscall.

8-) Child1 sleeps for ten seconds.

9-) Parent process creates a new child by using fork syscall.

10-) Child1 sleeps for ten seconds.

11-) While children is sleeping parent prints “proceeding” in a while loop each 2 seconds. While loop ends when the counter which counts the processes is 2.

12-) When the first child woke up, it opens FIFO files with open syscall and it reads array informations from FIFO file. For each syscall, it checks errors.

13-) Child1 sums up the random numbers and writes to the FIFO2 file by using write syscall.

14-) When everything finished, files are closed and allocated memories are freed.

15-) When the second child is woke up, it opens FIFO2 file. It reads, command, array and sum result from file.

16-) After multiplication and printing  result of summation, it closes file and frees allocated memory.

17-) When a child  exits, signal handler is called. In the waitpid, there is WNOHANG to prevent parent process to wait if there is no child. And according to status parameter, it prints exit statuses of processes. And for each process it increases counter.

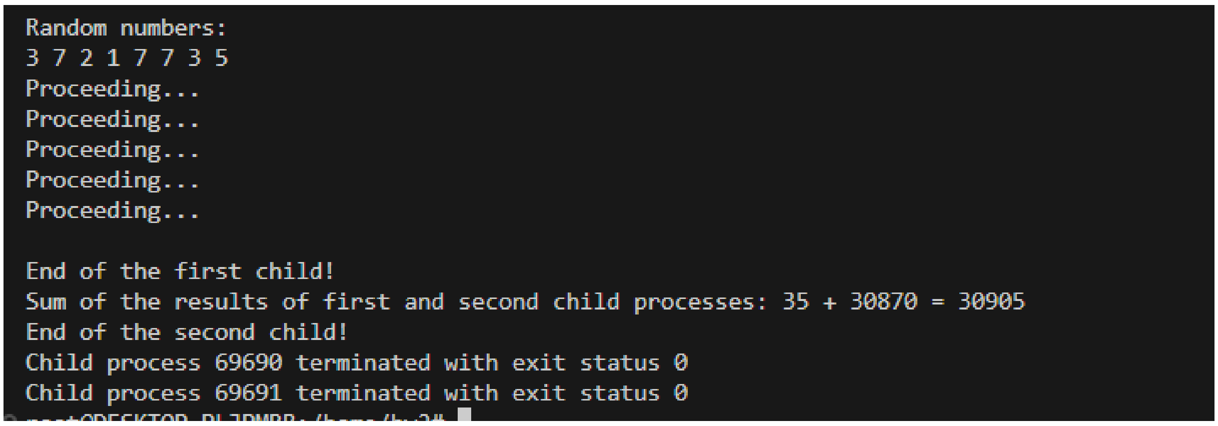
## BONUS PARTS:

In the signal handler function (sigchld_handler), when a child process is terminated it prints exit status according to waitpid’s status parameter.

To prevent zombie children, when a process is terminated, signal handler is called and it calls waitpid function in a while loop. So there is always a waitpid function to check whether to detect a child is terminated.

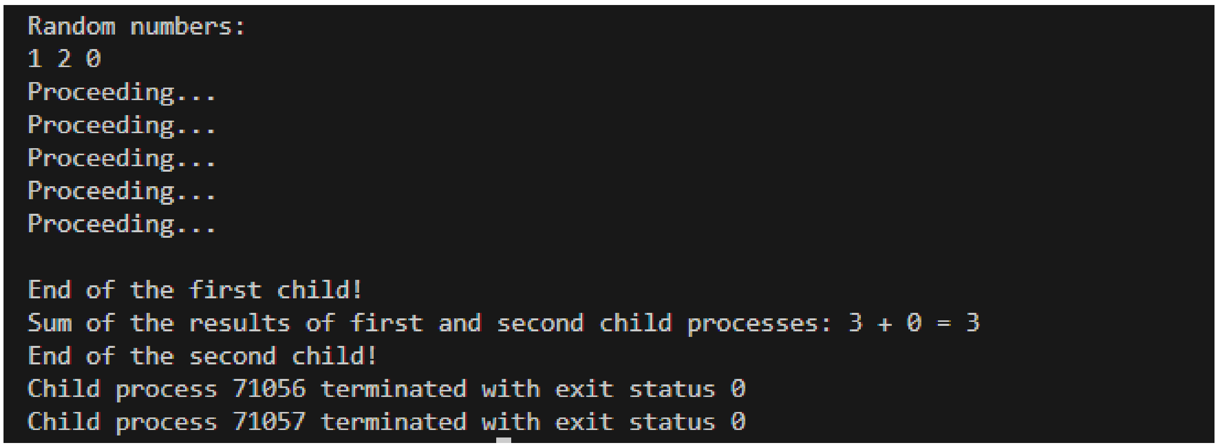
OUTPUT EXAMPLES:

With argument 8



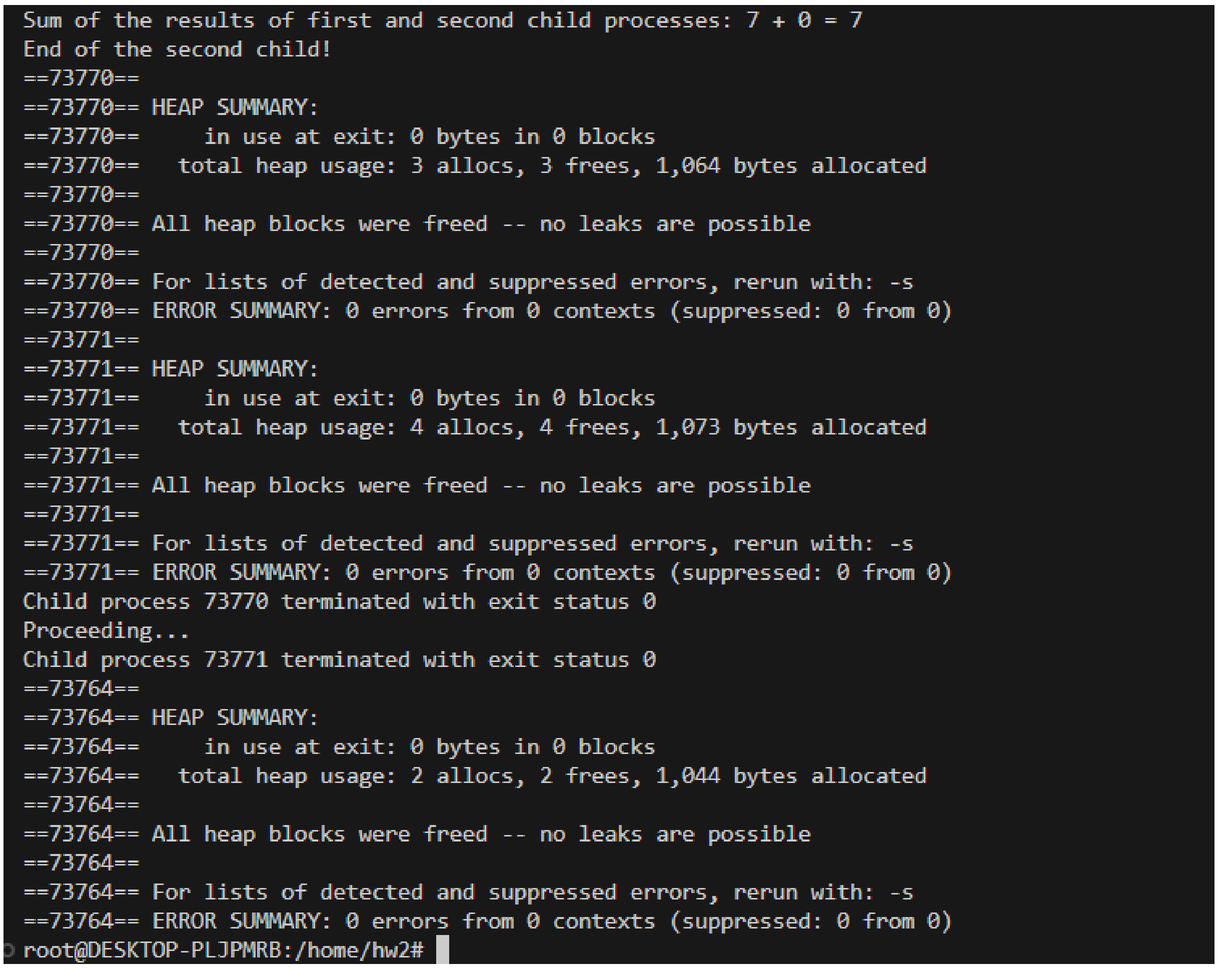
```
Random numbers:
3 7 2 1 7 7 3 5
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...

End of the first child!
Sum of the results of first and second child processes: 35 + 30870 = 30905
End of the second child!
Child process 69690 terminated with exit status 0
Child process 69691 terminated with exit status 0
```

With argument 3



```
Random numbers:
1 2 0
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...

End of the first child!
Sum of the results of first and second child processes: 3 + 0 = 3
End of the second child!
Child process 71056 terminated with exit status 0
Child process 71057 terminated with exit status 0
```

## Memory Leak Detection:



```
Sum of the results of first and second child processes: 7 + 0 = 7
End of the second child!
==73770==
==73770== HEAP SUMMARY:
==73770==     in use at exit: 0 bytes in 0 blocks
==73770==   total heap usage: 3 allocs, 3 frees, 1,064 bytes allocated
==73770==
==73770== All heap blocks were freed -- no leaks are possible
==73770==
==73770== For lists of detected and suppressed errors, rerun with: -s
==73770== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==73771==
==73771== HEAP SUMMARY:
==73771==     in use at exit: 0 bytes in 0 blocks
==73771==   total heap usage: 4 allocs, 4 frees, 1,073 bytes allocated
==73771==
==73771== All heap blocks were freed -- no leaks are possible
==73771==
==73771== For lists of detected and suppressed errors, rerun with: -s
==73771== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Child process 73770 terminated with exit status 0
Proceeding...
Child process 73771 terminated with exit status 0
==73764==
==73764== HEAP SUMMARY:
==73764==     in use at exit: 0 bytes in 0 blocks
==73764==   total heap usage: 2 allocs, 2 frees, 1,044 bytes allocated
==73764==
==73764== All heap blocks were freed -- no leaks are possible
==73764==
==73764== For lists of detected and suppressed errors, rerun with: -s
==73764== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PLJPMRB:/home/hw2#
```