## ZADANIE 1.

***Kod:***

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
int pid = fork();
        if(pid==0){
                //dziecko
                printf("dziecko: ");
        }
        else{
                //rodzic
                printf("rodzic: ");
        }
printf("%d \n",getpid());

}
```

***Output:***

```
[03/03/23]seed@VM:~$ ./zad1
rodzic: 25733
dziecko: 25734
```

***Wywołanie strace:***

```
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fdd8040e810) = 25640
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
brk(NULL)                               = 0x55ef22eba000
brk(0x55ef22edb000)                     = 0x55ef22edb000
getpid()                                = 25639
write(1, "rodzic: 25639 \n", 15rodzic: 25639
)        = 15
exit_group(0)                           = ?
+++ exited with 0 +++
```

Wywołanie clone jest ważne, ponieważ odpowiada ono za utworzenie procesu potomnego. Powiązałbym je z linią kodu: pid = fork()
Wywołanie getpid jest ważne, ponieważ dzięki niemu jestem w stanie uzyskać numer PID procesu, który w programie chciałem wyświetlić.
Wywołanie write jest ważne, ponieważ wyświetla informacje na ekranie dzięki czemu jestem je w stanie zobaczyć.

### *Wywołanie strace -f*

```
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fe108d09810) = 3229
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
brk(NULL)                               = 0x560049e71000
brk(0x560049e92000)                     = 0x560049e92000
getpid()                                = 3228
write(1, "rodzic: 3228 \n", 14rodzic: 3228
)         = 14
exit_group(0)                           = ?
+++ exited with 0 +++
strace: Process 3229 attached
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
brk(NULL)                               = 0x560049e71000
brk(0x560049e92000)                     = 0x560049e92000
getpid()                                = 3229
write(1, "dziecko: 3229 \n", 15dziecko: 3229
)        = 15
exit_group(0)                           = ?
+++ exited with 0 +++
```

W przypadku wywołania strace -f mamy do czynienia z monitorowaniem wszystkich procesów utworzonych przez proces nadrzędny, w przeciwieństwie do samego strace który śledzi jedynie wywołania systemowe jednego procesu.

## ZADANIE 2.

1. Uruchomiłem program w terminalu. Program został wcześniej skompilowany z opcją -g.
2. Uruchomiłem drugi terminal w którym wywołałem komendę
   ps aux | grep zad1_2
   Przez co mogłem się dowiedzieć jaki PID ma proces w którym wykonuje się zapętlony program.
3. Użyłem gdb do „otwarcia" odnalezionego wcześniej procesu.

```
[03/03/23]seed@VM:~$ sudo gdb -p 26382
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/l
This is free software: you are free to change and redistribu
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
```

4. Znalazłem w help odpowiednią opcję do zmiany wartości zmiennej.
5. Wykonałem set variable a=0 dzięki czemu zmieniłem wartość zmiennej a na 0, a następnie c aby kontynuować działanie.

```
(gdb) set variable a=0
(gdb) c
Continuing.
[Inferior 1 (process 26382) exited normally]
(gdb)
```

6. W rezultacie warunek w pętli został spełniony co spowodowało wyjście z nieskończonego zapętlenia a tym samym zakończenia działania programu uruchomionego wcześniej w drugiej konsoli.

```
[03/03/23]seed@VM:~$ ./zad1_2
[03/03/23]seed@VM:~$
```

## ZADANIE 3.

```
└─$ sudo valgrind --leak-check=yes ./zad3
==6282== Memcheck, a memory error detector
==6282== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==6282== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==6282== Command: ./zad3
==6282==
==6282== Invalid write of size 1
==6282==    at 0x10918B: main (in /home/marcin/zad3)
==6282==  Address 0x4a5004a is 0 bytes after a block of size 10 alloc'd
==6282==    at 0x48407B4: malloc (vg_replace_malloc.c:381)
==6282==    by 0x10915A: main (in /home/marcin/zad3)
==6282==
==6282==
==6282== HEAP SUMMARY:
==6282==     in use at exit: 30 bytes in 2 blocks
==6282==   total heap usage: 2 allocs, 0 frees, 30 bytes allocated
==6282==
==6282== 10 bytes in 1 blocks are definitely lost in loss record 1 of 2
==6282==    at 0x48407B4: malloc (vg_replace_malloc.c:381)
==6282==    by 0x10915A: main (in /home/marcin/zad3)
==6282==
==6282== 20 bytes in 1 blocks are definitely lost in loss record 2 of 2
==6282==    at 0x48407B4: malloc (vg_replace_malloc.c:381)
==6282==    by 0x109168: main (in /home/marcin/zad3)
==6282==
==6282== LEAK SUMMARY:
==6282==    definitely lost: 30 bytes in 2 blocks
==6282==    indirectly lost: 0 bytes in 0 blocks
==6282==      possibly lost: 0 bytes in 0 blocks
==6282==    still reachable: 0 bytes in 0 blocks
==6282==         suppressed: 0 bytes in 0 blocks
==6282==
==6282== For lists of detected and suppressed errors, rerun with: -s
==6282== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

Program wykrył 3 błędy związane z pamięcią.
Pierwszy oznacza zapisanie wartości na adres poza tablicą, zgodnie z programem tablica ma 10 elementów czyli indeksowanie kończy się na 9 nie jest więc możliwe wpisanie wartości 2 na indeks 10.
Kolejne dwa błędy związane są z brakiem zwolnienia pamięci zaalokowanej przez p1 i p2.

Przykładowa poprawa:

```c
// zad3.c
#include <stdlib.h>
#include <string.h>
int main() {
    unsigned char *p1 = malloc(10*sizeof(unsigned char));
    unsigned char *p2 = malloc(20*sizeof(unsigned char));
    memset(p1, 1, 10);
    p1[9] = 2;
    free(p1);
    free(p2);
    return 0;
}
```

Zmieniłem indeks na 1 mniejszy dzięki czemu program mieści się w zakresie. Dodatkowo zwalniam pamięć dla p1 i p2.

```
┌──(marcin㉿kali)-[~]
└─$ sudo valgrind --tool=memcheck --leak-check=yes ./zad3
==6493== Memcheck, a memory error detector
==6493== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==6493== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==6493== Command: ./zad3
==6493==
==6493==
==6493== HEAP SUMMARY:
==6493==     in use at exit: 0 bytes in 0 blocks
==6493==   total heap usage: 2 allocs, 2 frees, 30 bytes allocated
==6493==
==6493== All heap blocks were freed -- no leaks are possible
==6493==
==6493== For lists of detected and suppressed errors, rerun with: -s
==6493== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Poprawnym rozwiązaniem może być też:

```c
// zad3.c
#include <stdlib.h>
#include <string.h>
int main() {
    unsigned char *p1 = malloc(11*sizeof(unsigned char));
    unsigned char *p2 = malloc(20*sizeof(unsigned char));
    memset(p1, 1, 10);
    p1[10] = 2;
    free(p1);
    free(p2);
    return 0;
}
```

Gdzie zwiększam alokowaną pamięć p1 na 11 bajtów i podobnie jak wcześniej zwalniam pamięć zarówno p1 jak i p2. Oba rozwiązania dają takie samo