

Aufgabenblatt 8

Julian Bertol

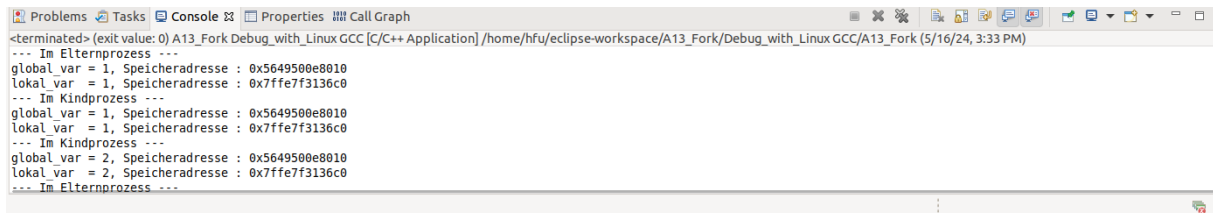
--Alle Code Dateien sind auf Felix hochgeladen!

Run A13_Fork:

Target:

```
pi@target085:~$ /tmp/remotetest/A13_Fork;exit
--- Im Elternprozess ---
global_var = 1, Speicheradresse : 0x21034
lokal_var = 1, Speicheradresse : 0x7eb56568
--- Im Kindprozess ---
global_var = 1, Speicheradresse : 0x21034
lokal_var = 1, Speicheradresse : 0x7eb56568
```

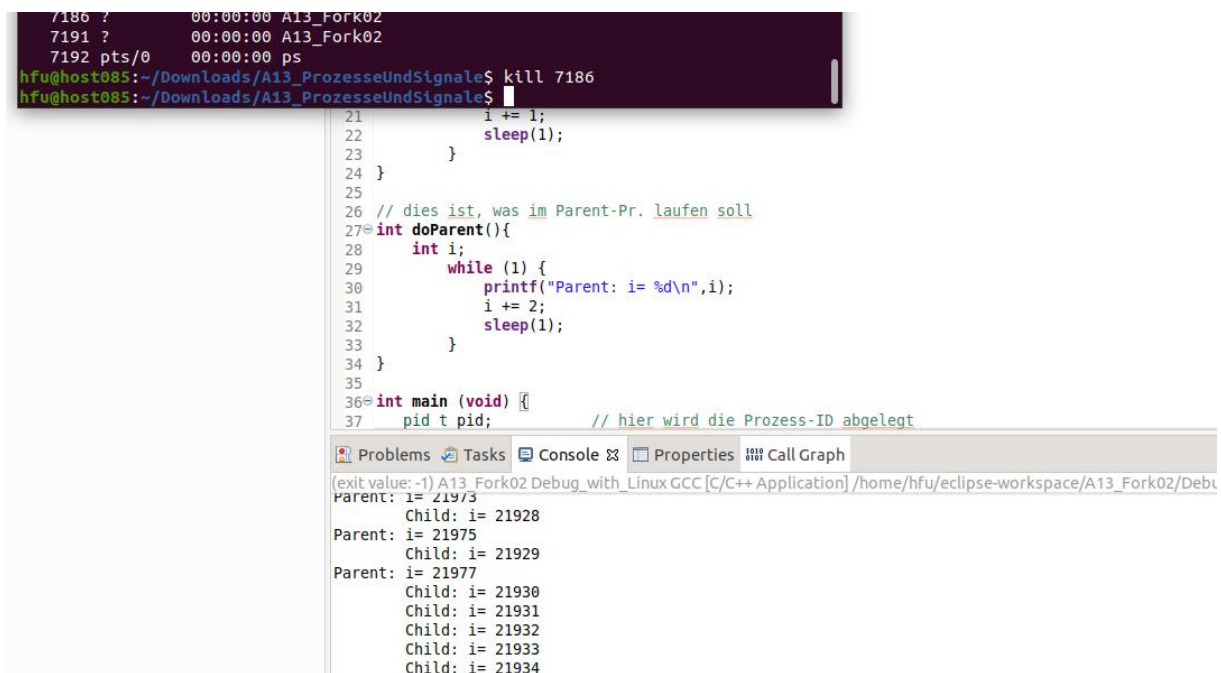
Host:



```
<terminated> (exit value: 0) A13_Fork Debug_with_Linux GCC [C/C++ Application] /home/hfu/eclipse-workspace/A13_Fork/Debug_with_Linux GCC/A13_Fork (5/16/24, 3:33 PM)
... Im Elternprozess ...
global_var = 1, Speicheradresse : 0x5649500e0010
lokal_var = 1, Speicheradresse : 0x7ffe7f3136c0
... Im Kindprozess ...
global_var = 1, Speicheradresse : 0x5649500e0010
lokal_var = 1, Speicheradresse : 0x7ffe7f3136c0
... Im Kindprozess ...
global_var = 2, Speicheradresse : 0x5649500e0010
lokal_var = 2, Speicheradresse : 0x7ffe7f3136c0
... Im Elternprozess ...
```

Die Laufvariablen sind unterschiedlich, weil die Variable i keinen Wert zugewiesen bekommen hat. Dies kann man beheben, indem man i einen Wert zuweist.

Mit ps -e sieht man alle laufenden Prozesse und die dazugehörige Prozess id. Mit kill <ID> kann man diese Prozesse beenden.



```
7186 ?      00:00:00 A13_Fork02
7191 ?      00:00:00 A13_Fork02
7192 pts/0   00:00:00 ps
hfu@host085:~/Downloads/A13_ProzesseUndSignale$ kill 7186
hfu@host085:~/Downloads/A13_ProzesseUndSignale$
21      i += 1;
22      sleep(1);
23  }
24  }
25
26 // dies ist, was im Parent-Pr. laufen soll
27 int doParent(){
28     int i;
29     while (1) {
30         printf("Parent: i= %d\n",i);
31         i += 2;
32         sleep(1);
33     }
34 }
35
36 int main (void) {
37     pid_t pid; // hier wird die Prozess-ID abgelegt
    ...
    Parent: i= 21973
    Child: i= 21928
    Parent: i= 21975
    Child: i= 21929
    Parent: i= 21977
    Child: i= 21930
    Child: i= 21931
    Child: i= 21932
    Child: i= 21933
    Child: i= 21934
```

```
7204 ?      00:00:00 A13_Fork02
7209 ?      00:00:00 A13_Fork02
7210 pts/0   00:00:00 ps
hfu@host085:~/Downloads/A13_ProzesseUndSignale$ kill 7209
hfu@host085:~/Downloads/A13_ProzesseUndSignale$
```

```
21         i += 1;
22         sleep(1);
23     }
24 }
25
26 // dies ist, was im Parent-Pr. laufen soll
27 int doParent(){
28     int i;
29     while (1) {
30         printf("Parent: i= %d\n",i);
31         i += 2;
32         sleep(1);
33     }
34 }
35
36 int main (void) {
37     pid_t pid; // hier wird die Prozess-ID abgelegt
```

Problems Tasks Console Properties Call Graph

A13_Fork02 Debug with Linux GCC [C/C++ Application]

Child: i= 21990

Parent: i= 21995
Child: i= 21991

Parent: i= 21997
Child: i= 21992

Parent: i= 21999
Child: i= 21993

Parent: i= 22001
Parent: i= 22003
Parent: i= 22005

Sig_handler:

```
void sig_handler(int signo)
{
    if (signo == SIGUSR1){
        printf("received SIGUSR1\n");
        i = 0;
    }
    else if (signo == SIGINT) {
        printf("received SIGINT\n");
        printf("restart again\n");
        i *= 2;
    }
    else if (signo == SIGUSR2){
        printf("received SIGUSR2\n");
        i = -i;
    }
    else if (signo == SIGKILL)
        printf("received SIGKILL\n");
    else if (signo == SIGSTOP)
        printf("received SIGSTOP\n");
}
```

```

int main(void) {
    printf("Hello World!!!\n"); /* prints !!!Hello World!!! */
    int pid= getpid();

    if (signal(SIGUSR1, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGUSR1\n");
    if (signal(SIGINT, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGINT\n");
    if (signal(SIGKILL, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGKILL\n");
    if (signal(SIGSTOP, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGSTOP\n");
    if (signal(SIGUSR2, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGSTOP\n");

    for (i=0; i<1000; i++){
        printf("PID= %d, i= %d\n",pid,i);
        sleep (1);
    }

    return EXIT_SUCCESS;
}

```

Erklärung zur Aufgabe 4:

Es gibt einen Sighandler der bei manchen Befehlen die PID überprüft und dadurch checkt, ob dieser Befehl an den Child oder Parent Prozess gesendet werden soll. Mir ist durchaus bewusst, dass man dies auch mit 2 Signalhandler lösen könnte, aber auf diese Idee bin ich zu spät gekommen und bin bei Variante 1 geblieben.