

CSC 2400 – Creating and Destroying Processes in Unix

Exercise 0 – Printing Own Process Identifier

pid.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main ()
{
    int id;

    id = getpid();
    printf("\n My identifier is ID = [%d]\n", id);
    sleep(10);
    return 0;
}
```

```
*****
Compile:          gcc -o pid pid.c
Run:              ./pid &
List Processes:   ps
*****
```

Output:

Exercise 1 – Creating New Process

fork1.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork1()
{
    int ret;

    ret = fork();
    if (ret == 0)
        printf("\n [%d] Hello from child", getpid());
    else
        printf("\n [%d] Hello from parent", getpid());
}

int main ()
{
    fork1();
    return 0;
}
```

```
*****
Compile:      gcc -o fork1 fork1.c
Run:          ./fork1
*****
```

Output:

Exercise 2 – Separate Parent and Child Spaces

fork2.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork2()
{
    int ret;
    int x = 10;

    ret = fork();
    if (ret == 0)
        printf("\n [%d] Child has x = %d\n", getpid(), ++x);
    else
        printf("\n [%d] Parent has x = %d\n", getpid(), --x);
}

int main ()
{
    fork2();
    return 0;
}
```

```
*****
Compile:      gcc -o fork2 fork2.c
Run:         ./fork2
*****
```

Output:

Exercise 3 – Parent and Child Continue fork-ing

fork3.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork3()
{
    printf("\n [%d] L0 \n", getpid());
    fork();
    printf("\n [%d] L1 \n", getpid());
    fork();
    printf("\n [%d] Bye \n", getpid());
}

int main ()
{
    fork3();
    return 0;
}
```

```
*****
Compile:      gcc -o fork3 fork3.c
Run:         ./fork3
*****
```

Output:

Exercise 4 – Parent Continues fork-ing

fork4.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork4()
{
    printf("\n [%d] L0 \n", getpid());
    if (fork() != 0)
    {
        printf("\n [%d] L1 \n", getpid());
        if (fork() != 0)
        {
            printf("\n [%d] L2 \n", getpid());
            fork();
        }
    }
    printf("\n [%d] Bye \n", getpid());
}

int main ()
{
    fork4();
    return 0;
}
```

```
*****
Compile:      gcc -o fork4 fork4.c
Run:         ./fork4
*****
```

Output:

Exercise 5 – Child Continues fork-ing

fork5.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork5()
{
    printf("\n [%d] L0 \n", getpid());
    if (fork() == 0)
    {
        printf("\n [%d] L1 \n", getpid());
        if (fork() == 0)
        {
            printf("\n [%d] L2 \n", getpid());
            fork();
        }
    }
    printf("\n [%d] Bye \n", getpid());
}

int main ()
{
    fork5();
    return 0;
}
```

```
*****
Compile:      gcc -o fork5 fork5.c
Run:         ./fork5
*****
```

Output:

Exercise 6 – Non-terminating Parent with Zombie (Defunct) Child

fork6.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork6()
{
    int ret;
    ret = fork();

    if (ret == 0)
    {
        printf("\n [%d] Ending Child \n", getpid());
        exit(0);
    }
    else
    {
        printf("\n [%d] Running Parent \n", getpid());
        while(1);    /* infinite loop */
    }
}

int main ()
{
    fork6();
    return 0;
}
```

```
*****
Compile:          gcc -o fork6 fork6.c
Run:              ./fork6 &
List Processes:   ps
Kill Parent:      kill <pid>
*****
```

Exercise 7 – Non-terminating Child

fork7.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork7()
{
    int ret;
    ret = fork();

    if (ret == 0)
    {
        printf("\n [%d] Running Child \n", getpid());
        while(1);    /* infinite loop */
    }
    else
    {
        printf("\n [%d] Ending Parent \n", getpid());
        exit(0);
    }
}

int main ()
{
    fork7();
    return 0;
}
```

```
*****
Compile:          gcc -o fork7 fork7.c
Run:              ./fork7 &
List Processes:   ps
Kill Child:       kill <pid>
*****
```

Exercise 8 – Synchronizing Parent with Child

fork8.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork8()
{
    int ret;
    ret = fork();

    if (ret == 0)
    {
        printf("\n [%d] Running Child \n", getpid());
        sleep(1);
        printf("\n [%d] Ending Child \n", getpid());
    }
    else
    {
        printf("\n [%d] Waiting Parent \n", getpid());
        wait(NULL);
        printf("\n [%d] Ending Parent \n", getpid());
    }
}

int main ()
{
    fork8();
    return 0;
}
```

```
*****
Compile:          gcc -o fork8 fork8.c
Run:              ./fork8 &
List Processes:   ps
Repeat List:      ps
*****
```