Tutorial 2

Question 1

1. fork --- wait

	C code:
00	`int main( ) {`
01	`    //This is process P`
02	`    if ( fork() == 0 ){`
03	`        //This is process Q`
04	`        if ( fork() == 0 ) {`
05	`            //This is process R`
06	`            ......`
07	`            return 0;`
08	`        }`
09	`        <Point α>`
10	`    }`
11	`    wait(NULL); <Point β>`
12	
13	`    return 0;`
14	`}`

Behaviour

Process Q ***always*** terminate before P.

Process R can terminate at any time w.r.t. P and Q.

1. fork --- wait

	C code:
00	`int main( ) {`
01	`    //This is process P`
02	`    if ( fork() == 0 ){`
03	`        //This is process Q`
04	`        if ( fork() == 0 ) {`
05	`            //This is process R`
06	`            ......`
07	`            return 0;`
08	`        }`
09	`        wait(NULL); <Point α>`
10	`    }`
11	`     <Point β>`
12	
13	`    return 0;`
14	`}`

Behaviour

Process Q ***always*** terminate before P.

Process R can terminate at any time w.r.t. P and Q.

1. fork --- wait

	C code:
00	`int main( ) {`
01	`    //This is process P`
02	`    if ( fork() == 0 ){`
03	`        //This is process Q`
04	`        if ( fork() == 0 ) {`
05	`            //This is process R`
06	`            ......`
07	`            return 0;`
08	`        }`
09	`        execl(valid executable....); <Point α>`
10	`    }`
11	`    wait(NULL); <Point β>`
12	
13	`    return 0;`
14	`}`

Behaviour
Process Q ***always*** terminate before P. Process R can terminate at any time w.r.t. P and Q.

1. fork --- wait

	C code:
00	`int main( ) {`
01	`    //This is process P`
02	`    if ( fork() == 0 ){`
03	`        //This is process Q`
04	`        if ( fork() == 0 ) {`
05	`            //This is process R`
06	`            ......`
07	`            return 0;`
08	`        }`
09	`        wait(NULL); <Point α>`
10	`    }`
11	`    wait(NULL); <Point β>`
12	
13	`    return 0;`
14	`}`

Behaviour

Process P **never terminates**.

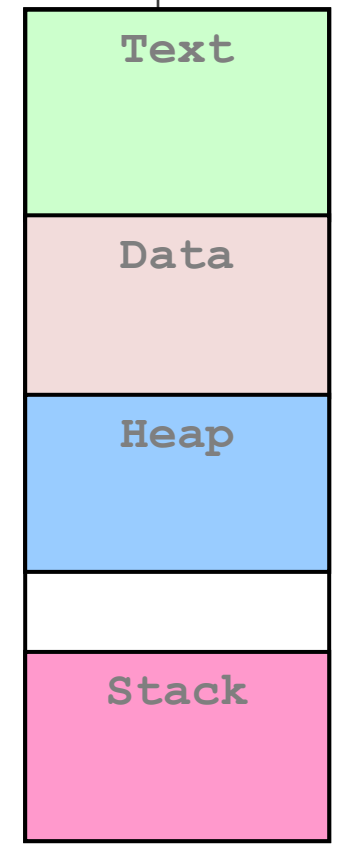
Tutorial 2

Question 2

a. dataX, dataY, region pointed by dataZptr

C code:

```
int dataX = 100;

int main( )
{
    pid_t childPID;

    int dataY = 200;
    int* dataZptr = (int*) malloc(sizeof(int));
```



Memory Space of a Process

b. Memory space after fork()?

```
PID[550761] | X = 100 | Y = 200 | Z = 300 |
*PID[550761] | X = 100 | Y = 200 | Z = 300 |
#PID[550761] | X = 101 | Y = 202 | Z = 303 |
**PID[550761] | X = 101 | Y = 202 | Z = 303 |
##PID[550761] | X = 102 | Y = 204 | Z = 306 |
*PID[550762] | X = 100 | Y = 200 | Z = 300 |
#PID[550762] | X = 101 | Y = 202 | Z = 303 |
**PID[550763] | X = 101 | Y = 202 | Z = 303 |
##PID[550763] | X = 102 | Y = 204 | Z = 306 |
**PID[550762] | X = 101 | Y = 202 | Z = 303 |
##PID[550762] | X = 102 | Y = 204 | Z = 306 |
**PID[550764] | X = 101 | Y = 202 | Z = 303 |
##PID[550764] | X = 102 | Y = 204 | Z = 306 |
```

C code:

```
int dataX = 100;

int main( )
{
    pid_t childPID;

    int dataY = 200;
    int* dataZptr = (int*) malloc(sizeof(int));
    *dataZptr = 300;

    printf("PID[%d] | X = %d | Y = %d | Z = %d |\n",
           getpid(),  dataX, dataY, *dataZptr);

    childPID = fork();
    printf("*PID[%d] | X = %d | Y = %d | Z = %d |\n",
           getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("#PID[%d] | X = %d | Y = %d | Z = %d |\n",
           getpid(),  dataX, dataY, *dataZptr);
```

C code:

```
int dataX = 100;

int main( )
{
    pid_t childPID;

    int dataY = 200;
    int* dataZptr = (int*) malloc(sizeof(int));
    *dataZptr = 300;

    printf("PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    childPID = fork();
    printf("*PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("#PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);
```

First

Second

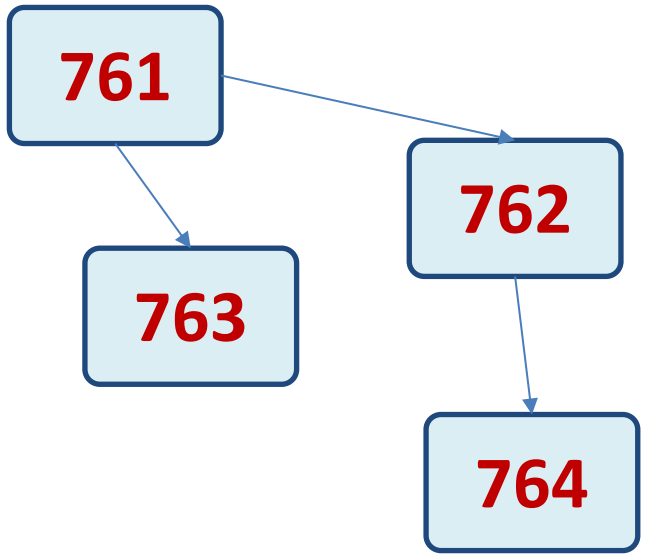
Code Insertion Point

```
    childPID = fork();
    printf("**PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("##PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    return 0;
}
```

Third

c. Process Tree

```
PID[550761] | X = 100 | Y = 200 | Z = 300 |
*PID[550761] | X = 100 | Y = 200 | Z = 300 |
#PID[550761] | X = 101 | Y = 202 | Z = 303 |
**PID[550761] | X = 101 | Y = 202 | Z = 303 |
##PID[550761] | X = 102 | Y = 204 | Z = 306 |
*PID[550762] | X = 100 | Y = 200 | Z = 300 |
#PID[550762] | X = 101 | Y = 202 | Z = 303 |
**PID[550763] | X = 101 | Y = 202 | Z = 303 |
##PID[550763] | X = 102 | Y = 204 | Z = 306 |
**PID[550762] | X = 101 | Y = 202 | Z = 303 |
##PID[550762] | X = 102 | Y = 204 | Z = 306 |
**PID[550764] | X = 101 | Y = 202 | Z = 303 |
##PID[550764] | X = 102 | Y = 204 | Z = 306 |
```



d, e: Message Ordering

C code:

```
int dataX = 100;

int main( )
{
    pid_t childPID;

    int dataY = 200;
    int* dataZptr = (int*) malloc(sizeof(int));
    *dataZptr = 300;

    printf("PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    childPID = fork();
    printf("*PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("#PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    childPID = fork();
    printf("**PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("##PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    return 0;
}
```

First

Second

Code Insertion Point

Third

```
PID[550761] | X = 100 | Y = 200 | Z = 300 |
*PID[550761] | X = 100 | Y = 200 | Z = 300 |
#PID[550761] | X = 101 | Y = 202 | Z = 303 |
**PID[550761] | X = 101 | Y = 202 | Z = 303 |
##PID[550761] | X = 102 | Y = 204 | Z = 306 |
*PID[550762] | X = 100 | Y = 200 | Z = 300 |
#PID[550762] | X = 101 | Y = 202 | Z = 303 |
**PID[550763] | X = 101 | Y = 202 | Z = 303 |
##PID[550763] | X = 102 | Y = 204 | Z = 306 |
**PID[550762] | X = 101 | Y = 202 | Z = 303 |
##PID[550762] | X = 102 | Y = 204 | Z = 306 |
**PID[550764] | X = 101 | Y = 202 | Z = 303 |
##PID[550764] | X = 102 | Y = 204 | Z = 306 |
```

f: Sleepy Child?

C code:

```
int dataX = 100;

int main( )
{
    pid_t childPID;

    int dataY = 200;
    int* dataZptr = (int*) malloc(sizeof(int));
    *dataZptr = 300;

    printf("PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    childPID = fork();
    printf("*PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("#PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    childPID = fork();
    printf("**PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("##PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    return 0;
}
```

First

Second

Code Insertion Point

Third

```
if (childPID == 0){
    sleep(5);
}
```

```
PID[550761] | X = 100 | Y = 200 | Z = 300 |
*PID[550761] | X = 100 | Y = 200 | Z = 300 |
#PID[550761] | X = 101 | Y = 202 | Z = 303 |
**PID[550761] | X = 101 | Y = 202 | Z = 303 |
##PID[550761] | X = 102 | Y = 204 | Z = 306 |
*PID[550762] | X = 100 | Y = 200 | Z = 300 |
#PID[550762] | X = 101 | Y = 202 | Z = 303 |
**PID[550763] | X = 101 | Y = 202 | Z = 303 |
##PID[550763] | X = 102 | Y = 204 | Z = 306 |
**PID[550762] | X = 101 | Y = 202 | Z = 303 |
##PID[550762] | X = 102 | Y = 204 | Z = 306 |
**PID[550764] | X = 101 | Y = 202 | Z = 303 |
##PID[550764] | X = 102 | Y = 204 | Z = 306 |
```

g: No child left behind?

C code:

```
int dataX = 100;

int main( )
{
    pid_t childPID;

    int dataY = 200;
    int* dataZptr = (int*) malloc(sizeof(int));
    *dataZptr = 300;

    printf("PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    childPID = fork();
    printf("*PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("#PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    childPID = fork();
    printf("**PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    dataX += 1;
    dataY += 2;
    (*dataZptr) += 3;
    printf("##PID[%d] | X = %d | Y = %d | Z = %d |\n",
            getpid(),  dataX, dataY, *dataZptr);

    return 0;
}
```

First

Second

Code Insertion Point

Third

```
if (childPID != 0){
    wait(NULL);
}
```

```
PID[550761] | X = 100 | Y = 200 | Z = 300 |
*PID[550761] | X = 100 | Y = 200 | Z = 300 |
#PID[550761] | X = 101 | Y = 202 | Z = 303 |
**PID[550761] | X = 101 | Y = 202 | Z = 303 |
##PID[550761] | X = 102 | Y = 204 | Z = 306 |
*PID[550762] | X = 100 | Y = 200 | Z = 300 |
#PID[550762] | X = 101 | Y = 202 | Z = 303 |
**PID[550763] | X = 101 | Y = 202 | Z = 303 |
##PID[550763] | X = 102 | Y = 204 | Z = 306 |
**PID[550762] | X = 101 | Y = 202 | Z = 303 |
##PID[550762] | X = 102 | Y = 204 | Z = 306 |
**PID[550764] | X = 101 | Y = 202 | Z = 303 |
##PID[550764] | X = 102 | Y = 204 | Z = 306 |
```

Tutorial 2

Question 3

```c
int main()
{
    int userInput, childPid, childResult;
    //Since largest number is 10 digits, a 12 characters string is more
    //than enough
    char cStringExample[12];

    scanf("%d", &userInput);

    childPid = fork();

    if (childPid != 0 ){
        wait( &childResult);
        printf("%d has %d prime factors\n", userInput,
                WEXITSTATUS(childResult));

    } else {
        //Easy way to convert a number into a string
        sprintf(cStringExample, "%d", userInput);

        execl("./PF", "PF", cStringExample, NULL);
    }

}
```

```c
int main( int argc, char* argv[])
{
    int nFactor = 0, userInput, factor;

    //Convert string to number
    userInput = atoi( argv[1] );

    nFactor = 0;
    factor = 2;

    //quick hack to get the number of prime factors
    // only for positive integer
    while (userInput > 1){
        if (userInput % factor == 0){
            userInput /= factor;
            nFactor++;
        } else {
            factor++;
        }
    }

    return nFactor;
}
```

```c
int main()
{
    int i, j, userInput[9], nInput, childPid[9], childResult, pid;
    char cStringExample[12];
    scanf("%d", &nInput);

    for (i = 0; i < nInput; i++){
        scanf("%d", &userInput[i]);

        childPid[i] = fork();
        if (childPid[i] == 0){
            sprintf(cStringExample, "%d", userInput[i]);
            execl("./PF", "PF", cStringExample, NULL);
            return 0; // Redundant. Everything from here downwards
                      // is replaced by PF in the child.
        }
    }
    for (i = 0; i < nInput; i++){
        pid = wait( &childResult );

        //match pid with child pid
        for (j = 0; j < nInput; j++){
            if (pid == childPid[j])
                break;
        }
          // We use the official WEXITSTATUS macro to ensure portability.
        printf("%d has %d prime factors\n", userInput[j],
                    WEXITSTATUS(childresult));
    }
}
```