ANKARA UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT
BLM334/COM334 LAB1
SUPPLEMENTARY DOCUMENT

Processes

A.Compiling C program in Linux

```
//cTest.c
#include <stdio.h>
int main(int argc, char *argv[ ]){
        int i;
        for (i=0; i < argc; i++)
              printf("command line argument [%d] = %s \n",i, argv[i]);

        return 0;
}
```

defining the path that includes c compiler: export PATH=$PATH:/usr/local/bin
compiling program: gcc – o ctest ctest.c (gcc –o exefilename filename)
running program: ./ctest 10 tolga

B.Parent and Child Process

fork() is used for creating child processes, in this section we exercise some codes related to this subject

Ex. Simple fork use

```
//forkTest.c
#include <stdio.h>
main()
{
  puts("Begin fork test.");
  fork();
  puts("End fork test.");
}
```

Try this code wihout using fork() and explain the differences

Ex.Parent Process and Child Processes

```
//ParentChildTest.c
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
main()
{
            int pid;
            int status;

            printf("PARENT: My PID is %d\n", getpid());
            printf("PARENT: My parent's PID is %d\n", getppid());

            pid = fork();
```

```
    if(pid == 0){
        printf("CHILD: My PID is %d\n", getpid());
        printf("CHILD: My parent's PID is %d\n", getppid());
    }

    else{
        printf("PARENT: My PID is %d\n", getpid());
        printf("PARENT: My child's PID is %d\n", pid);
    }

    printf("1234567890\n");
    exit(0);
}
```

On the shell use “ps u” the lıst of current processes

Ex. PatentChildTest2

```
//ParentChildTest2.c
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

main()
{
    int pid;
    int status;

    printf("PARENT: My PID is %d\n", getpid());
    printf("PARENT: My parent's PID is %d\n", getppid());

    pid = fork();
    if(pid == 0){
        sleep(10);
        printf("CHILD: My PID is %d\n", getpid());
        printf("CHILD: My parent's PID is %d\n", getppid());
    }

    else{
        printf("PARENT: My PID is %d\n", getpid());
        printf("PARENT: My child's PID is %d\n", pid);
    }

    printf("1234567890\n");
    exit(0);
}
```

Ex.This example describes, the function of wait(&status) system call

```
//waitTest.c
#include <unistd.h>
#include <sys/types.h>
```

```
#include <sys/wait.h>

main()
{
        int pid;
        int status;

        printf("PARENT: My PID is %d\n", getpid());
        printf("PARENT: My parent's PID is %d\n", getppid());

        pid = fork();
        if(pid == 0){
                sleep(10);
                printf("CHILD: My PID is %d\n", getpid());
                printf("CHILD: My parent's PID is %d\n", getppid());
        }

        else{
                printf("PARENT: My PID is %d\n", getpid());
                printf("PARENT: My child's PID is %d\n", pid);
                wait(&status);
                printf("PARENT: Child is done with status %d\n", status);
        }

        printf("1234567890\n");
        exit(0);
}
```

Ex:An example of command line argument with fork() system call

```
//procinterleave.c
#include <stdio.h>

main(int argc, char *argv[])
{
  int i, limit;

  if (argc == 1)
    {
    fputs("Error", stderr);
    exit(1);
    }
  limit = atoi(argv[1]);
  fork();
  for(i=1;i <= limit; i++)
    printf("%d \n",i);
}
```

Ec. The exec family of functions replaces the current process image with a new process image. Give the command “man exec” to investigate.

```
//Exec1.c
```

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

main()
{
        execl("/bin/ls", "ls", "/", 0);
        printf("Exec finished\n");
}
```

Ex.Exec1 used as a child

```
//Exec2.c
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

main()
{
        int pid;

        if ((pid = fork()) == 0) {
                execl("/bin/ls", "ls", "/", 0);
        }
        else {
                wait(&pid);
                printf("Exec finished\n");
        }
}
```

Ex.

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

int main(int argc, char **argv)
{
   pid_t ret;
   ret = fork();
   if (ret == -1) {
        perror("fork returned -1");
        exit(EXIT_FAILURE);
   }
   printf("The value of ret is  %d !. Which is either parent's or child's process id (PID)'\n", 
ret);
   if (ret == 0) {
        pid_t mypid = getpid();
        printf("The child says, \"my pid is %d.\"'\n", mypid);
   } else {
        printf("Just became parent of %d.'\n", ret);
   }
   pid_t mypid = getpid();
   printf("pid %d says hello!'\n", mypid);
   return EXIT_SUCCESS;
}
```