# EXPERIMENT NO: 2 (a)

**AIM :** Design a c program to implement the multiprogramming memory management implementation of Fork() using System call.

# PROGRAM:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// make two process which run same

// program after this instruction

fork();

printf("Hello world!\n");

return 0;

}

# Code:

#include <stdio.h>

 #include <sys/types.h>

 #include <unistd.h>

 void forkexample()

 {

 // child process because return value zero

if (fork() == 0)

Printf("Hello from Child!\n");

 // parent process because return value non-zero.

else

printf("Hello from Parent!\n");

}

 int main()

 { forkexample();

 return 0;

 }

# EXPERIMENT NO: 2(b)

AIM : Design a C program to implement the multiprogramming memory management implementation of Exit() using system call

**CODE:**

#include <sys/types.h>

#include <sys/wait.h>

int main(void)

{

pid\_t pid = fork();

if ( pid == 0 )

{

exit(9999); //passing value more than 255

}

int status;

waitpid(pid, &status, 0);

if ( WIFEXITED(status) )

{

int exit\_status = WEXITSTATUS(status);

printf("Exit code: %d\n", exit\_status);

}

return 0;

}

# EXPERIMENT NO: 2 (c)

AIM: Design a C program to implement the multiprogramming memory management implementation of exec() using system call

#include<stdio.h>

#include<unistd.h>

int main()

{

   int i;

   printf("I am EXEC.c called by execvp() ");

   printf("\n");

   return 0;

}

//execDemo.c

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main()

{

       //A null terminated array of character

       //pointers

       char \*args[]={"EXEC",NULL};

       execvp(args[0],args);

              printf("Ending-----");

   return 0;

}

# EXPERIMENT NO: 2 (d)

AIM: Design a C program to implement the multiprogramming memory management implementation of wait() using system call

# CODE:

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

pid\_t cpid;

if (fork()== 0)

exit(0); /\* terminate child \*/

else

cpid = wait(NULL);

printf("Parent pid = %d\n", getpid());

printf("Child pid = %d\n", cpid);

return 0;

}

# CODE:

#include<stdio.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

if (fork()== 0)

printf("HC: hello from child\n");

else

{

printf("HP: hello from parent\n");

wait(NULL);

printf("CT: child has terminated\n");

}

printf("Bye\n");

return 0;

}