Objective: A program to display PID & PPID

Source Code:

```
#include<stdio.h>

#include<unistd.h>

int main()

{  int pid,ppid;

   pid=getpid();

   ppid=getppid();

   printf("The Process ID is %d",pid);

   printf("\nThe Parent Process ID is %d",ppid);

   return(0);

}
```

Analysis:

In this program, there are two header files:

- stdio.h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations
- unistd.h is the name of the header file that provides access to the POSIX operating system API.

In the main program, the value of pid and ppid is extracted using the function getpid() and getppid() respectively. The extracted values of pid and ppid is displayed on the screen with the help of printf function.

Objective: A program that uses the fork

Source code:

```
#include<stdio.h>

#include<unistd.h>

int main()

{  printf("This demonstrates the fork\n");

   fork();

   printf("Hello world\n");

}
```

Analysis:

In this program, there are two header files:

- stdio.h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations
- unistd.h is the name of the header file that provides access to the POSIX operating system API.

In the main program, we can see that a function called fork() is called. Here fork() is the name of the system call that the parent process uses to "divide" itself ("fork" into two identical processes). After calling fork(), the created child process is actually an exact copy of the parent - which would probably be of limited use - so it replaces itself with another process.

Objective: A program to show the use of forking

Source code:

```
#include<stdio.h>
#include<unistd.h>
int main()
{  int pid;
   printf("Hello world\n");
   printf("I am the parent process with ID %d\n",getpid());
   printf("Here I am before the use of forking\n");
   pid=fork();
   printf("Here I am just after forking\n");
   if(pid==0)
   printf("I am a child process with ID %d\n",getpid());
   else
   printf("I am the parent process with PID %d\n",getpid());
}
```

Analysis:

In this program, there are two header files:

- stdio.h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations
- unistd.h is the name of the header file that provides access to the POSIX operating system API.

Here how the fork works in a program has been illustrated.

Objective: A program to determine a parent process or a child process created by fork

Source Code:

```
#include<stdio.h>
#include<unistd.h>
int main()
{  int pid;
   printf("The main program PID is %d\n",(int) getpid());
   pid=fork();
   if(pid!=0)
   {  printf("\nThis is the parent process with ID %d",(int) getpid());
      printf("\nThe child process ID is %d",pid);
   }
   else
   printf("\nThis is the child process with ID %d",(int) getpid());
   return(0);
}
```

Analysis:

In this program, there are two header files:

- stdio.h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations.
- unistd.h is the name of the header file that provides access to the POSIX operating system API.

Here the use of fork creates a child process exactly a copy of parent process. The user is unknown about which one is parent process and which one is child process. This program helps to distinguish the parent process and child process.

Objective: Implementation of strict alternation

Source Code:

```
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
void *thread1f(void *arg); void *thread2f(void *arg);
int turn=1;
int main()
{  pthread_t thid1;
   pthread_t thid2;
   pthread_create(&thid1,NULL,&thread1f,NULL);
   pthread_create(&thid2,NULL,&thread2f,NULL);
   pthread_join(thid1,NULL);
   pthread_join(thid2,NULL);
   return 0;
}
void *thread1f(void *arg)
{  int c=0;
   while(c++<200)
   {  while(turn!=1);
      fputc('b',stderr);
      turn=0;
   }
}
```

```
void *thread2f(void *arg)
{  int b=0;
   while(b++<200)
   {  while(turn!=0);
      fputc('c',stderr);
      turn=1;
   }
}
```

Analysis:

In this program, there are three header files:

- stdlib.h is the header of the general purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others.
- unistd.h is the name of the header file that provides access to the POSIX operating system API.
- pthread.h is a POSIX standard for threads.

Here implementation of strict alternation is shown. The idea of strict alternation is twofold:

- At any given instant in time, only one process can be executing the function critical_region()
- The two processes take turns running critical_region() (i.e. once process 0 runs critical_region(), it must wait for process 1 to run critical_region() before being allowed to run it again).

Objective: Implementation of lock variables

Source code:

```
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
void *thread1f(void *arg);   void *thread2f(void *arg);
int lock=0;
int main()
{  pthread_t thid1;   pthread_t thid2;
   pthread_create(&thid1,NULL,&thread1f,NULL);
   pthread_create(&thid2,NULL,&thread2f,NULL);
   pthread_join(thid1,NULL);   pthread_join(thid2,NULL);
   return 0;
}
void *thread1f(void *arg)
{  int a=0;
   while(a++<20)
   {  while(lock!=0);
      if(lock==0)
      lock=1;
      fputc('b',stderr);
      lock=0;
   }
}
```

```
void *thread2f(void *arg)
{  int b=0;
   while(b++<20)
   {  while(lock!=0);
      if(lock==0)
      lock=1;
      fputc('a',stderr);
      lock=0;
   }
}
```

Analysis:

In this program, there are three header files:

- stdlib.h is the header of the general purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others.
- unistd.h is the name of the header file that provides access to the POSIX operating system API.
- pthread.h is a POSIX standard for threads.

Here implementation of lock variable is shown. Lock variable is a lock where the thread simply waits in a loop ("spins") repeatedly checking until the lock becomes available. Since the thread remains active but isn't performing a useful task, the use of such a lock is a kind of busy waiting. Once acquired, lock variables will usually be held until they are explicitly released, although in some implementations they may be automatically released if the thread being waited on (that which holds the lock) blocks, or "goes to sleep".

Objective: A program that guarantees the display of child process is ahead of that of parent process

Source code:

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

int main(void)

{  int pid,status;

   printf("Hello world\n");

   pid=fork()

   if(pid==-1)

   {  perror("bad fork");

      exit(1);

   }

   if(pid==0)

   printf("I am the child process\n");

   else

   {  wait(&status);

      printf("I am the parent process\n");

   }

}
```

Analysis:

In this program, there is only one header file sys/wait.h which is used for the declarations for waiting. The program is all about displaying the child process created by fork ahead of the parent process.

Objective: A program to show the orphan process

Source code:

```
#include<stdio.h>
#include<unistd.h>
int main(void)
{  int pid;
   printf("I am the original process with PID %d and PPID %d\n",getPID(),getPPID());
   pid=fork();
   if(pid!=0)
   printf("I am the parent with PID %d and PPID %d\n",getPID(),getPPID());
   else
   {  sleep(4);
      printf("I am the child with PID %d and PPID %d\n",getPID(),getPPID());
   }
   printf("PID %d terminates\n",getPID());
}
```

Analysis:

In this program, there are two header files:

- stdio.h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations.
- unistd.h is the name of the header file that provides access to the POSIX operating system API.

Orphan process is a process, whose parents are dead, that means parents are either terminated, killed or exited but the child process is still alive. In Linux/Unix like operating systems, as soon as parents of any process are dead, re-parenting occurs, automatically.

Objective: A program to show Peterson's solution

Source code:

```
#define FALSE 0

#define TRUE 1

#define N 2

int turn;

int interested[N];

void enter_region(int process)

{  int other;

   other=1-process;

   interested[process]=TRUE;

   turn=process;

   while(turn==process && interested[other]==TRUE)

}

void leace_region(int process)

{  interested[process]=FALSE;

}
```

Analysis:

Peterson's solution is a concurrent programming algorithm for mutual exclusion that allows two processes to share a single-use resource without conflict, using only shared memory for communication. The algorithm uses two variables, interested and turn. A interested[n] value of true indicates that the process wants to enter the critical section. The variable turn holds the ID of the process whose turn it is.