# Real Time Systems
# Assignment 1.3

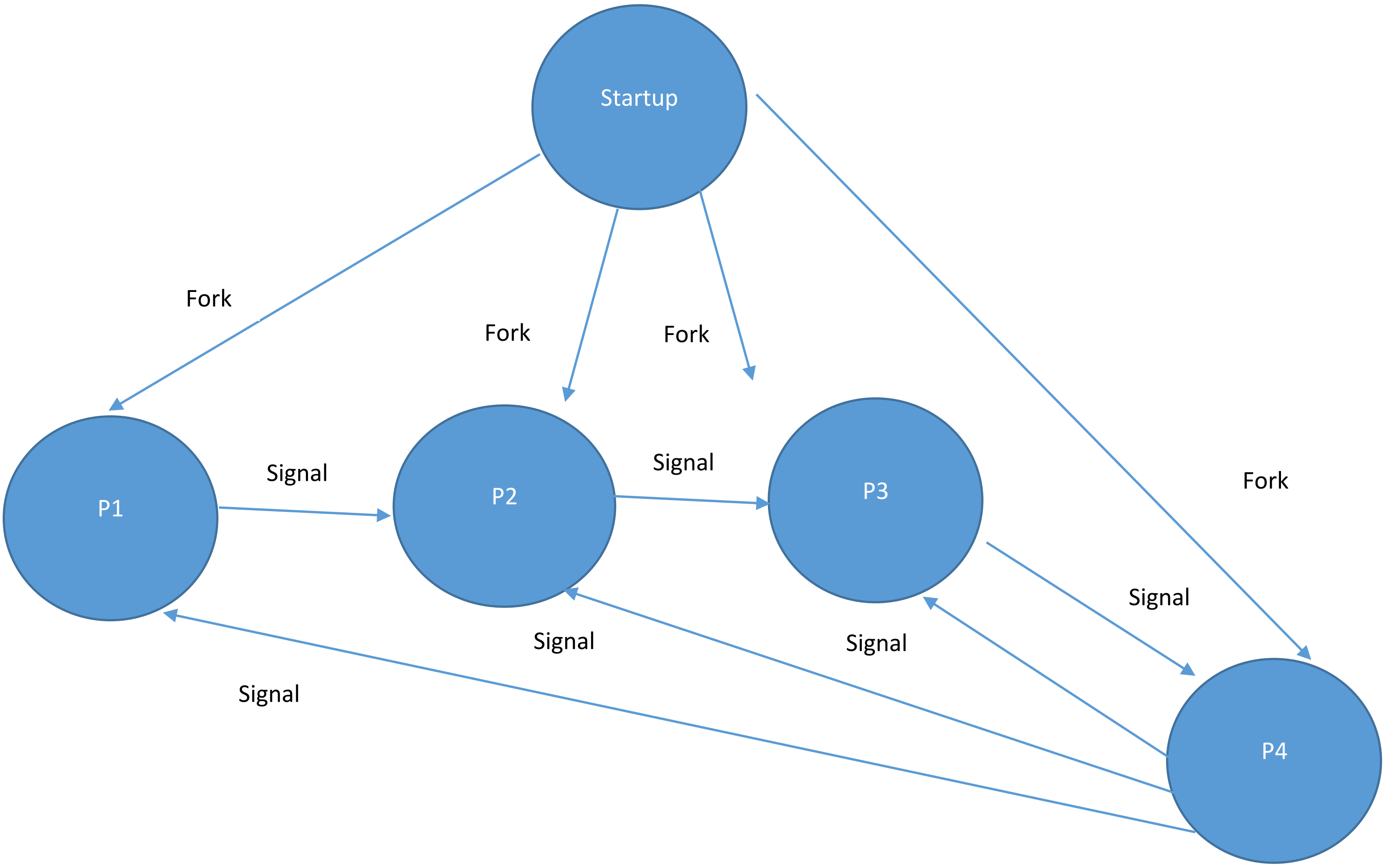
CIT CORK INSTITUTE OF TECHNOLOGY

INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Name: Michael O' Sullivan
Date:  2/12/16
Class: DCOM4

## Overview

For the assignment 1.3 of Real Time System we have to create a startup process that will fork four processes. The startup process forks each of the child process to initiate them. This is a diagram showing the processes. When startup forks the four process it will save the PID of each processes fork. Once all processes are forked, P1 will send a signal to P2, P2 will then send a signal to P3, P3 will send a signal to P4. P4 will then send a signal to P1, P2 and P3. Each process has access to he memory created in the startup containing all the PID of the process. This is used to send the signals to each process.



## Startup

This c file executes a fork, this fork starts processes P1, P2, P3 and P4. _os_exec() prepares the parameter and environment list before creating a process. _os_fork() creates a new process which becomes a child of the caller.  It sets up the new process' memory, MPU registers, and standard I/O paths. Before the processes are forked a _os_datamod is called to create a memory module to store all the PID's of the forked processes. _os_datmod() creates a data module with the specified attribute/ revision and clears the data portion of the module.  The module is created and entered into the system module directory.

```
#include <stdio.h>
#include <signal.h>
#include <module.h>
#include <types.h>
#include <errno.h>
#include <stdio.h>
#include <errno.h>
#include <process.h>
#include <dexec.h>
#include <types.h>
#include <string.h>
#include <modes.h>
#include <cglob.h>

#include "MemData.h"

#define MEMORY_NAME "CommonMem"


main(int argc , char * argv[], char **envp)
       {

       u_int16 attr_rev, type_lang;
       u_int16      perm,mem_size;
       mh_com mod_head;
       signal_code dummy_sig;
       u_int32 num_ticks;

       error_code err;

       process_id p1_pid;
       process_id p2_pid;
       process_id p3_pid;
       process_id p4_pid;


       status_code child_statusP1;
       status_code child_statusP2;
       status_code child_statusP3;
       status_code child_statusP4;


       char * process_argv[] = {
              "P1","P2", "P3", "P4"};
       struct MemData *CommonMem;

       mem_size = MEMORY_SIZE;
       type_lang = (MT_DATA << 8);
       attr_rev = (MA_REENT<<8);

       perm = MP_OWNER_READ | MP_OWNER_WRITE;

       if(errno = _os_datmod(MEMORY_NAME,mem_size, &attr_rev, &type_lang, perm,
              (void **)&CommonMem, (mh_data**)&mod_head) != 0)
              {
                  fprintf(stderr, "Error : Cannot create memory module\n");
            }
      else{
                  fprintf(stderr, "Successfully created memory module\n");
             }


      if (err = (_os_exec(_os_fork, 0 , 3 , process_argv[0], process_argv,
            envp,0,&p1_pid, 0,0) != 0))
            printf("Error1\n");

      if (err = (_os_exec(_os_fork, 0 , 3 , process_argv[1], process_argv,
            envp,0,&p2_pid, 0,0) != 0))
            printf("Error2\n");

      if (err = (_os_exec(_os_fork, 0 , 3 , process_argv[2], process_argv,
            envp,0,&p3_pid,0,0) != 0))
            printf("Error3\n");

      if (err = (_os_exec(_os_fork, 0 , 3 , process_argv[3], process_argv,
            envp,0,&p4_pid,0,0) != 0))
            printf("Error4\n");


      CommonMem -> PID[1] = p1_pid;
      CommonMem -> PID[2] = p2_pid;
      CommonMem -> PID[3] = p3_pid;
      CommonMem -> PID[4] = p4_pid;

    CommonMem ->MessageNumber = 222;

      /* wait for child P1 */
      if (err = (_os_wait(&p1_pid, &child_statusP1) != 0))
            printf("Error1\n");


      /* wait for child p2 */
      if (err = (_os_wait(&p2_pid, &child_statusP2) != 0))
            printf("Error2\n");


      /* wait for child  p3*/
      if (err = (_os_wait(&p3_pid, &child_statusP3) != 0))
            printf("Error3\n");


      /* wait for child  p3*/
      if (err = (_os_wait(&p4_pid, &child_statusP4) != 0))
            printf("Error4\n");


      while (1) {
             num_ticks = 500;
            _os_sleep(&num_ticks, &dummy_sig);
            }
}
```

**Status: 100%**

## P1

P1 once forked sleeps for 250 milliseconds so it can wait for all the PID’s to be stored in the data module. Once the delay is over, P1 creates a link to the memory module using  _os_link. P1 then sleeps for 200 milliseconds and sends a signal to P2 using the P2’s PID from the memory module.

```
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <modes.h>
#include <types.h>
#include <cglob.h>
#include <module.h>
#include "MemData.h"

#define MEMORY_NAME "CommonMem"

sig_handler(signal_code sig)
      {
      switch (sig)
            {
            case 400 : printf("P1: received a message -    Case 400\n");
                  break;
            case 500 : printf("P1: received a message from P4     Case 500\n");
            break;
         }
      _os_rte();
      }

main()
      {
            error_code err;
            u_int32 num_ticks;
            signal_code dummy_sig;
            u_int16 attr_rev, type_lang;
            u_int16 mem_size;
            mh_com mod_head;
            signal_code DummySignal;
            u_int32 SleepTime;
            char *ptrMemName;
            int i;
            int testPID;
            struct MemData *CommonMem;

            if ((err = _os_intercept(sig_handler, _glob_data)) != 0)
                  exit(err);

            SleepTime =  250;
            _os_sleep(&SleepTime, &DummySignal);
            type_lang = (MT_DATA << 8);
            attr_rev = (MA_REENT << 8);


            ptrMemName = MEMORY_NAME;

            errno = _os_link(&ptrMemName, (mh_com**)&mod_head, (void
                        **)&CommonMem, &type_lang, &attr_rev);

            num_ticks = 200;
            _os_sleep(&num_ticks, &dummy_sig);

            _os_send(CommonMem->PID[2], 400);

            while (1) {
                   num_ticks = 500;
                  _os_sleep(&num_ticks, &dummy_sig);
                  }
      }
```

**Status: 100%**

## P2

P2 once forked sleeps for 450 milliseconds so it can wait for all the PID’s to be stored in the data module. Once the delay is over, P2 creates a link to the memory module using _os_link. P2 then sleeps for 200 milliseconds and sends a signal to P3 using the P3’s PID from the memory module.

```
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <modes.h>
#include <types.h>
#include <cglob.h>
#include <module.h>

#include "MemData.h"

#define MEMORY_NAME "CommonMem"

sig_handler(signal_code sig)
       {
       switch (sig)
             {
             case 400 : printf("P2: received a signal -Case 400\n");
                    break;

             case 500 : printf("P2: received a message from P4 Case 500\n");
             break;
          }
       _os_rte();
       }

main()
       {

       error_code err;
       u_int32 num_ticks;
       signal_code dummy_sig;
       u_int16 attr_rev, type_lang;
       u_int16 mem_size;
       mh_com mod_head;
       signal_code DummySignal;
       u_int32 SleepTime;
       char *ptrMemName;
       int i;
       int testPID;
       struct MemData *CommonMem;


       type_lang = (MT_DATA << 8);
       attr_rev = (MA_REENT << 8);

       if ((err = _os_intercept(sig_handler, _glob_data)) != 0)
             exit(err);

       SleepTime =  450;
       _os_sleep(&SleepTime, &DummySignal);

       ptrMemName = MEMORY_NAME;


       errno = _os_link(&ptrMemName, (mh_com**)&mod_head, (void
       **)&CommonMem, &type_lang, &attr_rev);

       num_ticks = 200;
```

```
      _os_sleep(&num_ticks, &dummy_sig);

      _os_send(CommonMem -> PID[3], 400);

      while (1) {
              num_ticks = 500;
             _os_sleep(&num_ticks, &dummy_sig);
             }

}
```

**Status: 100%**

## P3

P3 once forked sleeps for 650milliseconds so it can wait for all the PID's to be stored in the data module. Once the delay is over, P3 creates a link to the memory module using _os_link. P3 then sleeps for 200 milliseconds and sends a signal to P4 using the P4's PID from the memory module.

```
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <modes.h>
#include <types.h>
#include <cglob.h>
#include <module.h>
#include "MemData.h"

#define MEMORY_NAME "CommonMem"

sig_handler(signal_code sig)
      {
      switch (sig)
            {
            case 400 : printf("P3: received a signal - Case 400\n");
                  break;

            case 500 : printf("P3: received a message from P4 Case 500\n");
            break;
         }
      _os_rte();
      }

main()
      {
            error_code err;
            u_int32 num_ticks;
            signal_code dummy_sig;
            u_int16 attr_rev, type_lang;
            u_int16 mem_size;
            mh_com mod_head;
            signal_code DummySignal;
            u_int32 SleepTime;
            char *ptrMemName;
            int i;
            int testPID;
            struct MemData *CommonMem;

            if ((err = _os_intercept(sig_handler, _glob_data)) != 0)
                  exit(err);
            SleepTime = 650;
            _os_sleep(&SleepTime, &DummySignal);
            type_lang = (MT_DATA << 8);
            attr_rev = (MA_REENT << 8);

            ptrMemName = MEMORY_NAME;


            errno = _os_link(&ptrMemName, (mh_com**)&mod_head, (void
                  **)&CommonMem, &type_lang, &attr_rev);

            num_ticks = 200;
            _os_sleep(&num_ticks, &dummy_sig);

            _os_send(CommonMem -> PID[4], 400);

            while (1)
            {
                   num_ticks = 500;
                  _os_sleep(&num_ticks, &dummy_sig);
            }


      }
```

**Status: 100%**

## P4

P4 once forked sleeps for 650milliseconds so it can wait for all the PID's to be stored in the data module. Once the delay is over, P4 creates a link to the memory module using _os_link. P4 then sleeps for 200 milliseconds and sends a signal to P1, P2 and P3 using their PID's from the memory module.

```
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <modes.h>
#include <types.h>
#include <cglob.h>
#include <module.h>
#include "MemData.h"

#define MEMORY_NAME "CommonMem"

sig_handler(signal_code sig)
      {
      switch (sig)
            {
            case 400 : printf("P4: received a signal -      Case 400\n");
                  break;
            case 500 : printf("P4: received a signal  Case 500\n");
                  break;
         }
      _os_rte();
      }
main()
      {
            error_code err;
            u_int32 num_ticks;
            signal_code dummy_sig;
            u_int16 attr_rev, type_lang;
            u_int16 mem_size;
            mh_com mod_head;
            signal_code DummySignal;
            u_int32 SleepTime;
            char *ptrMemName;
            int i;
            int testPID;
            struct MemData *CommonMem;

            if ((err = _os_intercept(sig_handler, _glob_data)) != 0)
                  exit(err);

            SleepTime = 850;
            _os_sleep(&SleepTime, &DummySignal);
            type_lang = (MT_DATA << 8);
            attr_rev = (MA_REENT << 8);

            ptrMemName = MEMORY_NAME;

            errno = _os_link(&ptrMemName, (mh_com**)&mod_head, (void
                  **)&CommonMem, &type_lang, &attr_rev);

            num_ticks = 200;
            _os_sleep(&num_ticks, &dummy_sig);


            _os_send(CommonMem -> PID[1], 500);
            _os_send(CommonMem -> PID[2], 500);
            _os_send(CommonMem -> PID[3], 500);


            while (1) {
                   num_ticks = 500;
                  _os_sleep(&num_ticks, &dummy_sig);
                  }

      }
```

**Status: 100%**

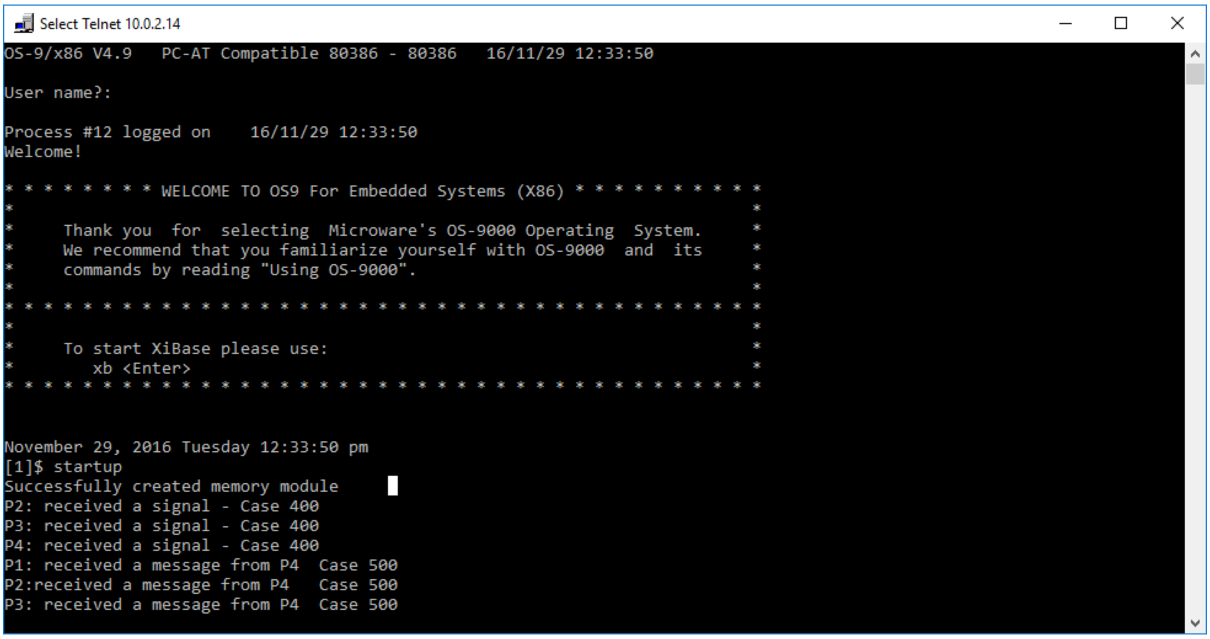
## Memory Module

The memory module has an array to store all the process ID's for the four processes.

```
#define MESS_SZ 30
struct MemData
      {
      int MessageNumber;
      int PID[5];
      }
#define MEMORY_SIZE sizeof(struct MemData)
```

**Status: 100%**

# Assignment 1.3 Running



```
OS-9/x86 V4.9   PC-AT Compatible 80386 - 80386   16/11/29 12:33:50

User name?:

Process #12 logged on    16/11/29 12:33:50
Welcome!

* * * * * * * * WELCOME TO OS9 For Embedded Systems (X86) * * * * * * * * * *
*                                                                            *
*     Thank you  for  selecting  Microware's OS-9000 Operating  System.      *
*     We recommend that you familiarize yourself with OS-9000  and  its      *
*     commands by reading "Using OS-9000".                                   *
*                                                                            *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                            *
*     To start XiBase please use:                                            *
*        xb <Enter>                                                          *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


November 29, 2016 Tuesday 12:33:50 pm
[1]$ startup
Successfully created memory module
P2: received a signal - Case 400
P3: received a signal - Case 400
P4: received a signal - Case 400
P1: received a message from P4  Case 500
P2:received a message from P4   Case 500
P3: received a message from P4  Case 500
```

**Status: 100%**

## Declaration of work

This report has been constructed and produced by Michael O' Sullivan. I declare that the report and its contents have been produced by Michael O Sullivan and is entirely his own work.

Mr. Michael O'Sullivan

R00077764

X_____Michael O Sullivan________________

Date: __02_____/ 11_____/ 16______