

Real Time Systems

Assignment 1.4



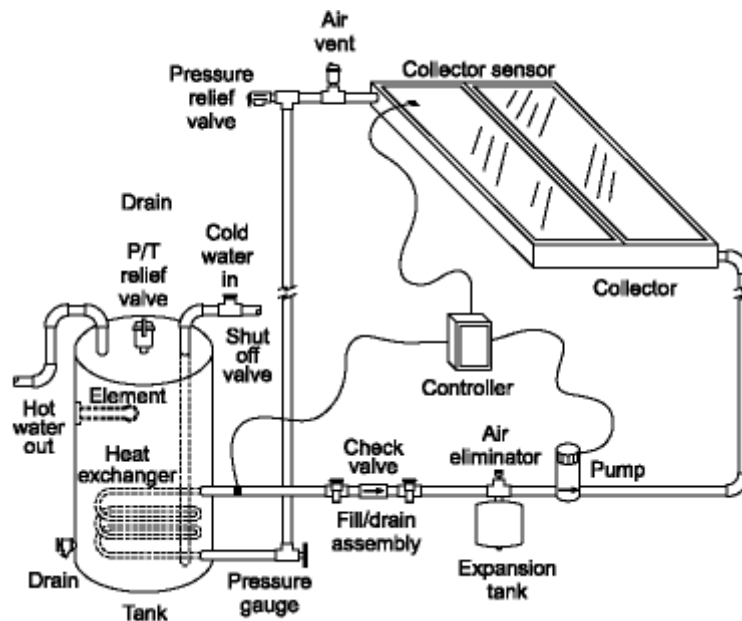
Name: Michael O' Sullivan

Date: 2/12/16

Class: DCOM4

Overview

For the assignment 1.4 of Real Time System I have to create a startup process that will fork seven processes. Each process is used build a solar heating system like the one illustrated below.



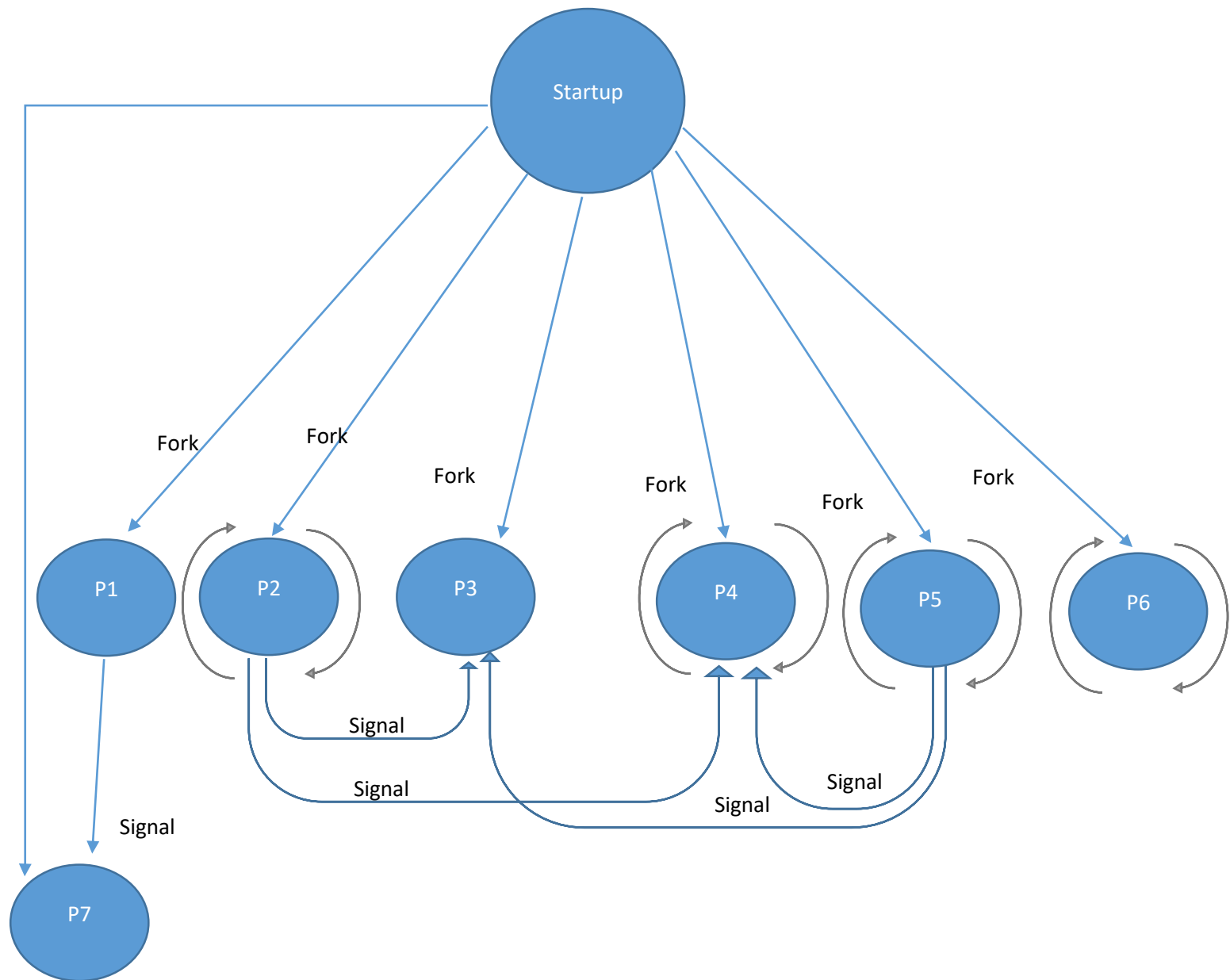
Each process has an individual job:

- 1) Will act as a sensor for the tank level.
- 2) This will act as a sensor for the air temperature on the outside. This is relevant as if its cold outside the solar panel will no produce any heat to heat the water.
- 3) This will turn on and off the pump if air outside is too cold or the water in the solar panel connector is cold as you don't want cold water entering the tank as it would lower the temperature inside the tank.
- 4) This will act as a sensor to increase/decrease the temperature of the tank if the pump is off.
- 5) This act as a sensor for the water temperature inside the tank. If the levels drop below a certain degrees it will send a signal to turn off the pump.
- 6) This process changes the tilt of the solar panel to obtain he best heating surface on the solar panel.
- 7) This wait for a signal from process one, if the water levels reach to high the valve will open to release some water, if the water levels reach to low the valve will close.

Three data modules will also be created

- 1) Store Processes ID in an array.
- 2) Store Solar Water Data.
- 3) Store Constant variables related to the solar heating system.

The startup process forks each of the child process to initiate them. This is a diagram showing the processes. When startup forks the seven process it will save the PID of each processes fork.



Startup

This c file executes a fork, this fork starts processes P1, P2, P3, P4, P5, P6 and P7. `_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 three memory module to store all the information about the water system. `_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. Startup will also create semaphore so only one process can access a file at one time, the other processes will have to join the queue. The semaphore event is created using `_os_ev_creat()`.

```
#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 <memory.h>
#include <svctbl.h>

#include "waterData.h"
#include "MemData.h"
#include "constantValues.h"

#define MEMORY_NAME "CommonMem"
#define MEMORY_WATER "WaterData"
#define MEMORY_WATER_CONSTANTS "WaterDataConstants"

#define EVENT_NAME "SemaEvent"
#define EVENT_WATER_DATA "SemaEventWater"

main(int argc , char * argv[], char **envp)
{
    signal_code DummySignal;
    u_int32 SleepValue;
    u_int16 attr_rev, type_lang;
    u_int16 perm, mem_size, mem_size_water, mem_size_constants;
    mh_com mod_head;
    u_int32 num_ticks;

    error_code err;

    process_id p1_pid;
    process_id p2_pid;
    process_id p3_pid;
    process_id p4_pid;
    process_id p5_pid;
    process_id p6_pid;
    process_id p7_pid;
```

```

status_code child_statusP1;
status_code child_statusP2;
status_code child_statusP3;
status_code child_statusP4;
status_code child_statusP5;
status_code child_statusP6;
status_code child_statusP7;

event_id ev_id;
event_id ev_id1;

char * process_argv[] = {
    "P1", "P2", "P3", "P4", "P5", "P6", "P7"};

struct MemData *CommonMem;
struct waterData *WaterData;
struct waterDataConstants *WaterDataConstants;

mem_size = MEMORY_SIZE;
mem_size_water = MEMORY_WATER_SIZE;
mem_size_constants = MEMORY_WATER_CONSTANTS_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(errno = _os_datmod(MEMORY_WATER,mem_size_water, &attr_rev, &type_lang,
    perm, (void **)&WaterData, (mh_data**)&mod_head) != 0)
{
    fprintf(stderr, "Error : Cannot create memory module
        MEMORY_WATER\n");
}
else{
    fprintf(stderr, "Successfully created memory module
        MEMORY_WATER\n");
}

if(errno = _os_datmod(MEMORY_WATER_CONSTANTS,mem_size_constants, &attr_rev,
    &type_lang, perm, (void **)&WaterDataConstants, (mh_data**)&mod_head) != 0)
{
    fprintf(stderr, "Error : Cannot create memory module
        MEMORY_WATER_CONSTANTS\n");
}
else{
    fprintf(stderr, "Successfully created memory module
        MEMORY_WATER_CONSTANTS\n");
}
}
/*Create Event*/
if((errno = _os_ev_creat(1, -1, perm, &ev_id, EVENT_NAME, 1, MEM_ANY)) != 0)
{
    printf("error getting access to event.");
    exit(errno);
}

```

```

        /*Create Event*/
        if((errno = _os_ev_creat(1, -1, perm, &ev_id1, EVENT_WATER_DATA, 1,
            MEM_ANY)) != 0)
        {
            printf("error getting access to event.");
            exit(errno);
        }

        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");

        if (err = (_os_exec(_os_fork, 0 , 3 , process_argv[4], process_argv,
            envp,0,&p5_pid, 0,0) != 0))
            printf("Error2\n");

        if (err = (_os_exec(_os_fork, 0 , 3 , process_argv[5], process_argv,
            envp,0,&p6_pid, 0,0) != 0))
            printf("Error3\n");

        if (err = (_os_exec(_os_fork, 0 , 3 , process_argv[6], process_argv,
            envp,0,&p7_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 -> PID[5] = p5_pid;
CommonMem -> PID[6] = p6_pid;
CommonMem -> PID[7] = p7_pid;

WaterData -> MessageNumbers = 0;
WaterData -> tankLevelHight = 55;
WaterData -> tankValve = 1;          /*Valve open*/
WaterData -> tankTemp = 65;
WaterData -> waterCollectionTemp = 75;
WaterData -> pumpSpeed = 12;

WaterDataConstants -> MessageNumbersConstants = 0;
WaterDataConstants -> tankMaxHeight = 90;
WaterDataConstants -> pumpSpeedMax = 20;

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 p4*/
if (err = (_os_wait(&p4_pid, &child_statusP4) != 0))
    printf("Error4\n");

/* wait for child p5 */
if (err = (_os_wait(&p5_pid, &child_statusP5) != 0))
    printf("Error5\n");

/* wait for child p6*/
if (err = (_os_wait(&p6_pid, &child_statusP6) != 0))
    printf("Error6\n");

/* wait for child p7*/
if (err = (_os_wait(&p7_pid, &child_statusP7) != 0))
    printf("Error7\n");

SleepValue = 0;
while(1)
    _os_sleep(&SleepValue, &DummySignal);

}

```

Status: 100%

P1

P1 once created sleeps for 100 milliseconds. An alarm cycle is created for every 3 seconds. It opens the water data module to get data such as if the valve is open, If the valve is open the then the water level will decrease if the valve closed the water will increase. P1 connected to the data module using `_os_link()`. Inside the alarm there is an `os_ev_link` to the event semaphore created in the startup.

```

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

#define EVENT_NAME "SemaEvent"
#define EVENT_WATER_DATA "SemaEventWater"

#define MEMORY_NAME "CommonMem"
#define MEMORY_WATER "WaterData"

```

```

main()
{
    signal_code DummySignal;
    u_int32 SleepTime;
    mh_com mod_head;
    event_id ev_id;
    u_int32 value;
    signal_code signal;
    char *ptrMemName, *ptrMemWater;
    error_code err;
    u_int16 attr_rev, type_lang;

    signal_code ReceivedSignal;
    u_int32 SleepValue;

    alarm_id MyAlarm;
    signal_code WakeupSignal;

    u_int32 TimeToDelay;

    struct MemData *CommonMem;
    struct waterData *WaterData;

    ptrMemName = MEMORY_NAME;
    ptrMemWater = MEMORY_WATER;

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

    SleepTime = 100;
    _os_sleep(&SleepTime, &DummySignal);

    TimeToDelay = 3000;
    WakeupSignal = 1;

    if ((errno = _os_alarm_cycle(&MyAlarm, WakeupSignal, TimeToDelay)) != 0)
    {
        printf("error creating alarm\n");
    }

    SleepValue = 0; /* Infinite loop, sleep forever */

    while (1){
        _os_sleep(&SleepValue, &ReceivedSignal);
        if(ReceivedSignal ==0){

            if((errno = _os_ev_link(EVENT_WATER_DATA, &ev_id)) != 0)
            {
                printf("Event 'EVENT_WATER_DATA' not linked?\n");
                exit(errno);
            }

            if((errno = _os_ev_wait(ev_id, &value,&signal,1,1))!= 0)
            {
                printf("P2: Waiting on event error\n");
                exit(0);
            }

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

```



```

        /*increment tank level to make it work like a sensor
        data being read in
        *if the valve is open(1) decrement the water level
        *value water is being released
        *if the valve is closed(0) increment the water level
        value as water is not being released

        */

        if(WaterData -> tankValve == 1){
            WaterData -> tankLevelHight = WaterData ->
            tankLevelHight -1;
        }
        if(WaterData -> tankValve == 0)
        {
            WaterData -> tankLevelHight = WaterData ->
            tankLevelHight+1;
        }

        printf ("P1: Tank Level Hight = %d \n", WaterData ->
            tankLevelHight);

        if(WaterData -> tankLevelHight > 90)
        {
            _os_send(CommonMem->PID[7], 400);
        }
        if(WaterData -> tankLevelHight < 50){
            _os_send(CommonMem->PID[7], 500);
        }

        /*increment message number*/
        WaterData -> MessageNumbers = WaterData ->
            MessageNumbers +1 ;

        /* Exit from the critical section, therefore the
        semaphore event is realesed. */
        if((errno = _os_ev_signal(ev_id, &value, 0)) != 0)
        {
            printf("Signalling event error\n");
            exit(errno);
        }

        /* Unlink from the semaphore evet. */
        if((errno = _os_ev_unlink(ev_id)) != 0)
        {
            printf("Event unlink error\n");
            exit(errno);
        }
    }
else
    printf("P1: The signal value which caused the alarm is %d\n",
        ReceivedSginal);

}

}

```

Status: 100%

P2

P2 once forked sleeps for 100 milliseconds so it can wait data from the startup to be loaded into the data modules. Once the delay is over, P2 creates an alarm for every 20 seconds. This act as a sensor to describe the outside air conditions. These temperatures are taken from

<http://www.met.ie/climate/daily-data.asp> and the figures are taken from 16/5/16 at Cork Airport.

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

#define EVENT_NAME "SemaEvent"
#define EVENT_WATER_DATA "SemaEventWater"

#define MEMORY_NAME "CommonMem"
#define MEMORY_WATER "WaterData"

main()
{
    signal_code DummySignal;
    u_int32 SleepTime;
    mh_com mod_head;
    event_id ev_id;
    u_int32 value;
    signal_code signal;
    char *ptrMemName, *ptrMemWater;
    error_code err;
    u_int16 attr_rev, type_lang;

    signal_code ReceivedSignal;
    u_int32 SleepValue;

    alarm_id MyAlarm;
    signal_code WakeupSignal;

    u_int32 TimeToDelay;
    u_int32 i;

    struct MemData *CommonMem;
    struct waterData *WaterData;

    /*24 values to resemble hourly air temperatures(deg C)
    *Starting at 1 am and finishing at 23.59
    *Figures are taken from Cork Airport on 16/5/2016
    *http://www.met.ie/climate/daily-data.asp
    */
    int reads[] = {11, 9, 9, 10, 9, 8, 9, 10, 11, 12, 13, 12, 13, 14,
                   15, 15, 15, 15, 16, 15, 14, 13, 11, 10, 9, 8};
    i = 0;
```

```

ptrMemName = MEMORY_NAME;
ptrMemWater = MEMORY_WATER;

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

SleepTime = 100;
_os_sleep(&SleepTime, &DummySignal);

/*3600000 is an hour
TimeToDelay = 3600000;
For the demo I will leave it at every 20 seconds.
*/

TimeToDelay = 20000;
WakeupSignal = 1;

if ((errno = _os_alarm_cycle(&MyAlarm, WakeupSignal, TimeToDelay)) != 0)
{
    printf("error creating alarm\n");
}

SleepValue = 0; /* Infinite loop, sleep forever */

while (1){
    _os_sleep(&SleepValue, &ReceivedSignal);
    if(ReceivedSignal ==0){

        if((errno = _os_ev_link(EVENT_WATER_DATA, &ev_id)) != 0)
        {
            printf("Event 'EVENT_WATER_DATA' not linked?\n");
            exit(errno);
        }

        if((errno = _os_ev_wait(ev_id,&value,&signal,1,1))!= 0)
        {
            printf("P2: Waiting on event error\n");
            exit(0);
        }

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

        printf("P2: Air temperture: %d\n", reads[i]);
        WaterData -> airTemp = reads[i] ;

        /*      If the air temp is less then 10 Deg C, turn off
        *      the solar heating and use gas/oil heating instead.
        *      The 10 Deg C is a fabricated figure to test the
        *      air temp data.
        */
        if(WaterData -> airTemp < 10){
            /*Start reducing the water temperture of the tank
            *due to it
            *not being heated by the solar system.
            *Also turn off pump as you dont want the cold
            *water circulating
            *around the system. This pump is just for pumping
            *water from the solar panels.
            *I will keep measuring the tank level as the
            *oil/gas system could still be
            *used.
            */

```

```

        _os_send(CommonMem->PID[3], 500); /*Turn off Pump*/
        _os_send(CommonMem->PID[4], 500); /*Reduce the
        water tempeture*/

    }

    if(WaterData -> airTemp >= 10){
        _os_send(CommonMem->PID[3], 400); /*Turn on Pump*/
        _os_send(CommonMem->PID[4], 400); /*Increase the
        water tempeture*/
    }

    /*increment message number*/
    WaterData -> MessageNumbers = WaterData ->
        MessageNumbers +1 ;

    /* Exit from the critical section, therefore the
    semaphore event is realeased. */
    if((errno = _os_ev_signal(ev_id, &value, 0)) != 0)
    {
        printf("Signalling event error\n");
        exit(errno);
    }

    /* Unlink from the semaphore evet. */
    if((errno = _os_ev_unlink(ev_id)) != 0)
    {
        printf("Event unlink error\n");
        exit(errno);
    }

    if(    i==23)
        i=0;
    else
        i= i +1;

}
else
    printf("P2: The signal value which caused the alarm is %d\n",
        ReceivedSginal);

}

}

```

Status: 100%

P3

P3 once forked sleeps for 100 milliseconds. P3 then waits for an interrupt. Once an interrupt occurs the sig_handle function is called. In the switch statement another function is called and is passed the signal code. Using the signal code, it can turn on or off the pump. The data is then written to the data module to say that the pump is either on or off.

```

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

```

```

#include "MemData.h"
#include "waterData.h"
#include "constantValues.h"

#define EVENT_NAME "SemaEvent"
#define EVENT_WATER_DATA "SemaEventWater"

#define MEMORY_NAME "CommonMem"
#define MEMORY_WATER "WaterData"
#define MEMORY_CONSTANTS "ConstantValues"

sig_handler(signal_code sig)
{
    switch (sig)
    {
        case 400 : printf("P3: Pump      ON\n");
                   chanageValveStatus(sig);
                   break;
        case 500 : printf("P3: Pump      OFF\n");
                   chanageValveStatus(sig);
                   break;
    }
    _os_rte();
}

chanageValveStatus(signal_code sig){
    mh_com mod_head;
    event_id ev_id;
    int finished;
    u_int32 value;
    signal_code signal;
    char *ptrMemName, *ptrMemWater, *ptrMemConstant;
    u_int16 attr_rev, type_lang;
    struct MemData *CommonMem;
    struct waterData *WaterData;
    struct waterDataConstants *ConstantValues;
    u_int32 TimeToDelay;
    u_int32 i;
    alarm_id MyAlarm;
    signal_code WakeupSignal;
    u_int32 SleepValue;
    signal_code ReceivedSignal;

    ptrMemName = MEMORY_NAME;
    ptrMemWater = MEMORY_WATER;
    ptrMemConstant = MEMORY_CONSTANTS;

    type_lang = (MT_DATA << 8);
    attr_rev = (MA_REENT << 8);
    if((errno = _os_ev_link(EVENT_WATER_DATA, &ev_id)) != 0)
    {
        printf("Event 'EVENT_WATER_DATA' not linked?\n");
        exit(errno);
    }

    if((errno = _os_ev_wait(ev_id, &value, &signal, 1, 1)) != 0)
    {
        printf("P3: Waiting on event error\n");
        exit(0);
    }

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

```

```

if(sig == 400)
{
    /*printf("P3: Pump running....\n");*/
    WaterData -> pumpSpeed = 50;

}

if(sig == 500)
{
    /*printf("P3: Pump stopped....\n");*/
    WaterData -> pumpSpeed = 0;
}

/*increment message number*/
WaterData -> MessageNumbers = WaterData -> MessageNumbers +1;

/* Exit from the critical section, therefore the semaphore event is
   released. */
if((errno = _os_ev_signal(ev_id, &value, 0)) != 0)
{
    printf("Signalling event error\n");
    exit(errno);
}

/* Unlink from the semaphore event. */
if((errno = _os_ev_unlink(ev_id)) != 0)
{
    printf("Event unlink error\n");
    exit(errno);
}
}

main()
{
    error_code err;
    signal_code DummySignal;
    u_int32 SleepTime;

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

    SleepTime = 100;
    _os_sleep(&SleepTime, &DummySignal);

    SleepTime = 0;
    while(1) {
        _os_sleep(&SleepTime, &DummySignal);
    }
}

```

Status: 100%

P4

P4 once forked sleeps for 100 milliseconds so it can wait data from the startup to be loaded into the data modules. Once the delay is over, P4 creates an alarm for every 2.5 seconds. This act as a sensor to produce the temperature of the tank. It takes in data such as in if the pump is running to find out if the water will be heating up or cooling. It uses the pump as the pump will be shut off if the conditions outside are not suitable to heat the water.

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

#define EVENT_NAME "SemaEvent"
#define EVENT_WATER_DATA "SemaEventWater"

#define MEMORY_NAME "CommonMem"
#define MEMORY_WATER "WaterData"

sig_handler(signal_code sig)
{
    switch (sig)
    {
        case 500 : printf("P4: Water cooling down...\n");
                   break;

        case 400 : printf("P4: Water heating up...\n");
                   break;
    }
    _os_rte();
}

main()
{
    signal_code DummySignal;
    u_int32 SleepTime;
    mh_com mod_head;
    event_id ev_id;
    u_int32 value;
    signal_code signal;
    char *ptrMemName, *ptrMemWater;
    error_code err;
    u_int16 attr_rev, type_lang;

    signal_code ReceivedSignal;
    u_int32 SleepValue;

    alarm_id MyAlarm;
    signal_code WakeupSignal;

    u_int32 TimeToDelay;
    u_int32 i;

    struct MemData *CommonMem;
    struct waterData *WaterData;

    ptrMemName = MEMORY_NAME;
    ptrMemWater = MEMORY_WATER;
```

```

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

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

SleepTime = 100;
_os_sleep(&SleepTime, &DummySignal);

TimeToDelay = 2500;
WakeupSignal = 1;

if ((errno = _os_alarm_cycle(&MyAlarm, WakeupSignal, TimeToDelay)) != 0)
{
    printf("P4: error creating alarm\n");
}

SleepValue = 0; /* Infinite loop, sleep forever */

while (1){
    _os_sleep(&SleepValue, &ReceivedSignal);
    if(ReceivedSignal == 0){

        if((errno = _os_ev_link(EVENT_WATER_DATA, &ev_id)) != 0)
        {
            printf("Event 'SemaEvent' not linked?\n");
            exit(errno);
        }

        if((errno = _os_ev_wait(ev_id, &value, &signal, 1, 1)) != 0)
        {
            printf("P4: Waiting on event error\n");
            exit(0);
        }

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

        if(WaterData -> pumpSpeed == 0){
            /*pump is shut off*/
            /*I dont want the water to go into a minus figure
            as it
            * it wouldnt happen in a real water tank in a
            house
            */
            if (WaterData -> tankTemp != 0 && WaterData
                -> pumpSpeed != 0){

                WaterData -> tankTemp = WaterData ->
                    tankTemp -1;
                printf ("P4: Water Tank Tempeture = %d C
                    \n", WaterData -> tankTemp);
            }
        }

        if(WaterData -> pumpSpeed > 0){/*pump is turned on*/
            if (WaterData -> tankTemp != 0 && WaterData ->
                pumpSpeed > 0){
                WaterData -> tankTemp = WaterData ->
                    tankTemp +1;
                printf ("P4: Water Tank Tempeture = %d C
                    \n", WaterData -> tankTemp);
            }
        }
    }
}

```



```

    }

}

/*increment message number*/
WaterData -> MessageNumbers = WaterData ->
MessageNumbers +1 ;

W

/* Exit from the critical section, therefore the
semaphore event is realesed. */
if((errno = _os_ev_signal(ev_id, &value, 0)) != 0)
{
    printf("Signalling event error\n");
    exit(errno);
}

/* Unlink from the semaphore evet. */
if((errno = _os_ev_unlink(ev_id)) != 0)
{
    printf("Event unlink error\n");
    exit(errno);
}
}
else
printf("P4: The signal value which caused the alarm is %d\n",
ReceivedSginal);
}

}
}

```

Status: 100%

P5

P5 once forked sleeps for 100 milliseconds so it can wait data from the startup to be loaded into the data modules. Once the delay is over, P5 creates an alarm for every 10seconds. This act as a sensor to produce the temperature of the collection tank in the solar panel. It takes in the temperature and if the temperature drops below a certain value a signal will be sent to P3 to turn off the pump so cold water isn't circulating in the system.

```

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

#define EVENT_NAME "SemaEvent"
#define EVENT_WATER_DATA "SemaEventWater"

#define MEMORY_NAME "CommonMem"
#define MEMORY_WATER "WaterData"

main()
{
    signal_code DummySignal;
    u_int32 SleepTime;
    mh_com mod_head;

```

```

event_id ev_id;
u_int32 value;
signal_code signal;
char *ptrMemName, *ptrMemWater;
error_code err;
u_int16 attr_rev, type_lang;
signal_code ReceivedSignal;
u_int32 SleepValue;
alarm_id MyAlarm;
signal_code WakeupSignal;
u_int32 TimeToDelay;
u_int32 i;

struct MemData *CommonMem;
struct waterData *WaterData;

/*24 values to resemble water tempeture (deg C)
*during a 24 hour day, the data is collected on the hour.
*These figures a fabricated for demonstration purposes
*/
int reads[] = {40, 39, 39, 40, 39, 38, 39, 40, 41, 42, 45, 50, 53, 60,
67, 71, 75, 76, 76, 75, 64, 53, 41, 40, 39, 39};
i = 0;

ptrMemName = MEMORY_NAME;
ptrMemWater = MEMORY_WATER;

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

SleepTime = 100;
_os_sleep(&SleepTime, &DummySignal);

/*3600000 is an hour
TimeToDelay = 3600000;
For the demo I will leave it at every 20 seconds.
*/

TimeToDelay = 10000;
WakeupSignal = 1;

if ((errno = _os_alarm_cycle(&MyAlarm, WakeupSignal, TimeToDelay)) != 0)
{
    printf("error creating alarm\n");
}

SleepValue = 0; /* Infinite loop, sleep forever */

while (1){
    _os_sleep(&SleepValue, &ReceivedSignal);
    if(ReceivedSignal ==0){

        if((errno = _os_ev_link(EVENT_WATER_DATA, &ev_id)) != 0)
        {
            printf("Event 'SemaEvent' not linked?\n");
            exit(errno);
        }

        if((errno = _os_ev_wait(ev_id,&value,&signal,1,1)) != 0)
        {
            printf("P2: Waiting on event error\n");
            exit(0);
        }
    }
}

```

```

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

    printf("P5: Water collection temp:  %d\n", reads[i]);
    WaterData -> waterCollectionTemp = reads[i];

    /*If the water in the container on the solar panel
    goes below 40 deg C turn
    *the pump off.
    * the 40 Deg C is a fabricated figure to test the air
    temp data.
    */

    if(WaterData -> waterCollectionTemp < 40 && WaterData ->
        pumpSpeed != 0 ){
        /*Start reducing the water tempeture of the tank
        due to it
        * not being heated by the solar system.
        * Also turn off pump as you dont want the cold
        * water circulating
        * around the system. This pump is just for pumping
        * water from the solar panels.
        * It will keep measuring the tank level as the
        * oil/gas system could still be used.
        * Check if the pump is already off.
        */
        _os_send(CommonMem->PID[3], 500); /*Turn off Pump*/
        _os_send(CommonMem->PID[4], 500); /*Reduce the
            water tempeture*/
    }

    if(WaterData -> airTemp >= 40 && WaterData -> pumpSpeed
        > 0 ){
        _os_send(CommonMem->PID[3], 400); /*Turn on Pump*/
        _os_send(CommonMem->PID[4], 400); /*Increase the
            water tempeture*/
    }

    /*increment message number*/
    WaterData -> MessageNumbers = WaterData ->
    MessageNumbers +1 ;

    /* Exit from the critical section, therefore the
    semaphore event is realeased. */
    if((errno = _os_ev_signal(ev_id, &value, 0)) != 0)
    {
        printf("Signalling event error\n");
        exit(errno);
    }

    /* Unlink from the semaphore evet. */
    if((errno = _os_ev_unlink(ev_id)) != 0)
    {
        printf("Event unlink error\n");
        exit(errno);
    }

    if(    i==23)
        i=0;

```

```

        else
            i= i +1;
    }
    else
        printf("P5: The signal value which caused the alarm is %d\n",
            ReceivedSginal);

}

}

```

Status: 100%

P6

P6 once forked sleeps for 100 milliseconds so it can wait data from the startup to be loaded into the data modules. Once the delay is over, P6 creates an alarm for every 60 seconds. This acts as a sensor to produce a different tilt on the solar panel. The alarm will iterate through 12 value to resemble the 12 months of the year, the recommend tilts are available on

<http://solarelectricityhandbook.com/solar-angle-calculator.html>

The data modules are updated with the latest tilt value.

```

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

#define EVENT_NAME "SemaEvent"
#define EVENT_WATER_DATA "SemaEventWater"

#define MEMORY_NAME "CommonMem"
#define MEMORY_WATER "WaterData"

main()
{
    signal_code DummySignal;
    u_int32 SleepTime;
    mh_com mod_head;
    event_id ev_id;
    u_int32 value;
    signal_code signal;
    char *ptrMemName, *ptrMemWater;
    error_code err;
    u_int16 attr_rev, type_lang;

    signal_code ReceivedSginal;
    u_int32 SleepValue;

    alarm_id MyAlarm;
    signal_code WakeupSignal;

    u_int32 TimeToDelay;
    u_int32 i;

    struct MemData *CommonMem;
    struct waterData *WaterData;

```

```

/* These values will represent the angle of solar panel.
 * This will give the optimum angle to get the best out of the system.
 * The values are monthly based. The values are available on the url below.
 * http://solarelectricityhandbook.com/solar-angle-calculator.html
 * The values are for Cork.
 */
int reads[] = {22,30,38,46,54,62,54,46,38,30,22,14};
i = 0;

ptrMemName = MEMORY_NAME;
ptrMemWater = MEMORY_WATER;

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

SleepTime = 100;
_os_sleep(&SleepTime, &DummySignal);

/*2629746000 is 1 month in milliseconds
TimeToDelay = 2629746000;
For the demo I will leave it at every 60 seconds.
*/

TimeToDelay = 60000;
WakeupSignal = 1;

if ((errno = _os_alarm_cycle(&MyAlarm, WakeupSignal, TimeToDelay)) != 0)
{
    printf("error creating alarm\n");
}

SleepValue = 0; /* Infinite loop, sleep forever */

while (1){
    _os_sleep(&SleepValue, &ReceivedSignal);
    if(ReceivedSignal ==0){

        if((errno = _os_ev_link(EVENT_WATER_DATA, &ev_id)) != 0)
        {
            printf("P6: Event 'SemaEvent' not linked?\n");
            exit(errno);
        }

        if((errno = _os_ev_wait(ev_id,&value,&signal,1,1)) != 0)
        {
            printf("P6: Waiting on event error\n");
            exit(0);
        }

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

        WaterData -> solarPanelTilt = reads[i];

        printf("P6: Solar Panel Tilt:  %d degree angle\n",
            reads[i]);
        WaterData -> airTemp = reads[i] ;

        /*increment message number */

```

```

        WaterData -> MessageNumbers = WaterData ->
            MessageNumbers +1 ;

        /* Exit from the critical section, therefore the
        semaphore event is realeased. */
        if((errno = _os_ev_signal(ev_id, &value, 0)) != 0)
        {
            printf("Signalling event error\n");
            exit(errno);
        }

        /* Unlink from the semaphore evet. */
        if((errno = _os_ev_unlink(ev_id)) != 0)
        {
            printf("P6: Event unlink error\n");
            exit(errno);
        }

        if(    i==11)
            i=0;
        else
            i= i +1;
    }
    else
        printf("P6: The signal value which caused the alarm is %d\n",
            ReceivedSginal);

}

}

```

Status: 100%

P7

P7 once forked sleeps for 100 milliseconds. P7 then waits for an interrupt. Once an interrupt occurs the sig_handle function is called. In the switch statement another function is called and is passed the signal code. Using the signal code, it can open or close the water valve to stop the tank over filling. The data is then written to the data module to say that the valve is either on or off using binary values.

```

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

#define EVENT_NAME "SemaEvent"
#define EVENT_WATER_DATA "SemaEventWater"

#define MEMORY_NAME "CommonMem"
#define MEMORY_WATER "WaterData"

sig_handler(signal_code sig)
{
    switch (sig)
    {
        case 400 : printf("P7:Water Valve Open\n");
                    chanageValveStatus(sig);
    }
}

```

```

        break;
    case 500 : printf("P7:Water Valve Closed\n");
               chanageValveStatus(sig);
               break;
    }

    _os_rte();
}

chanageValveStatus(signal_code sig){
    mh_com mod_head;
    event_id ev_id;
    u_int32 value;
    signal_code signal;
    char *ptrMemName, *ptrMemWater;
    u_int16 attr_rev, type_lang;
    struct MemData *CommonMem;
    struct waterData *WaterData;

    ptrMemName = MEMORY_NAME;
    ptrMemWater = MEMORY_WATER;

    type_lang = (MT_DATA << 8);
    attr_rev = (MA_REENT << 8);
    if((errno = _os_ev_link(EVENT_WATER_DATA, &ev_id)) != 0)
    {
        printf("Event 'EVENT_WATER_DATA' not linked?\n");
        exit(errno);
    }

    if((errno = _os_ev_wait(ev_id, &value, &signal, 1, 1)) != 0)
    {
        printf("Waiting on event error\n");
        exit(0);
    }

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

    if(sig == 400)
    {
        WaterData -> tankValve = 1;
    }

    if(sig == 500)
    {
        WaterData -> tankValve = 0;
    }

    /*increment message number*/
    WaterData -> MessageNumbers = WaterData -> MessageNumbers + 1 ;

    /* Exit from the critical section, therefore the semaphore event is
    realeased. */
    if((errno = _os_ev_signal(ev_id, &value, 0)) != 0)
    {
        printf("Signalling event error\n");
        exit(errno);
    }

    /* Unlink from the semaphore evet. */
    if((errno = _os_ev_unlink(ev_id)) != 0)
    {
        printf("Event unlink error\n");
        exit(errno);
    }
}

```

```

}

main()
{
    error_code err;
    signal_code DummySignal;
    u_int32 SleepTime;

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

    SleepTime = 100;
    _os_sleep(&SleepTime, &DummySignal);

    SleepTime = 0;
    while(1) {
        _os_sleep(&SleepTime, &DummySignal);
    }
}

```

Status: 100%

Data Modules

MemData

```

#define MESS_SZ 30
struct MemData;

struct MemData
{
    int MessageNumber;
    int PID[5];
};

#define MEMORY_SIZE sizeof(struct MemData)

```

WaterData

```

#define MESS_Sze 30
struct waterData
{
    int MessageNumbers;
    int tankLevelHight;
    int tankValve;
    int tankTemp;
    int waterCollectionTemp;
    int solarPanelTilt;
    int pumpSpeed;
    int airTemp;
};

#define MEMORY_WATER_SIZE sizeof(struct waterData)

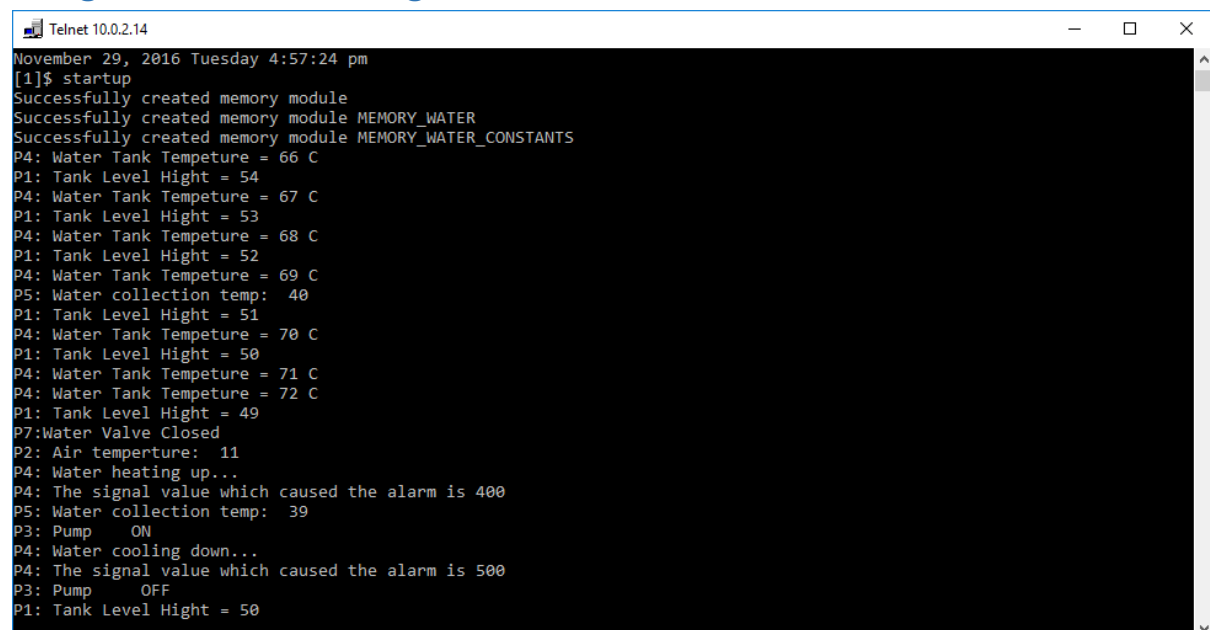
```


ConstantValues

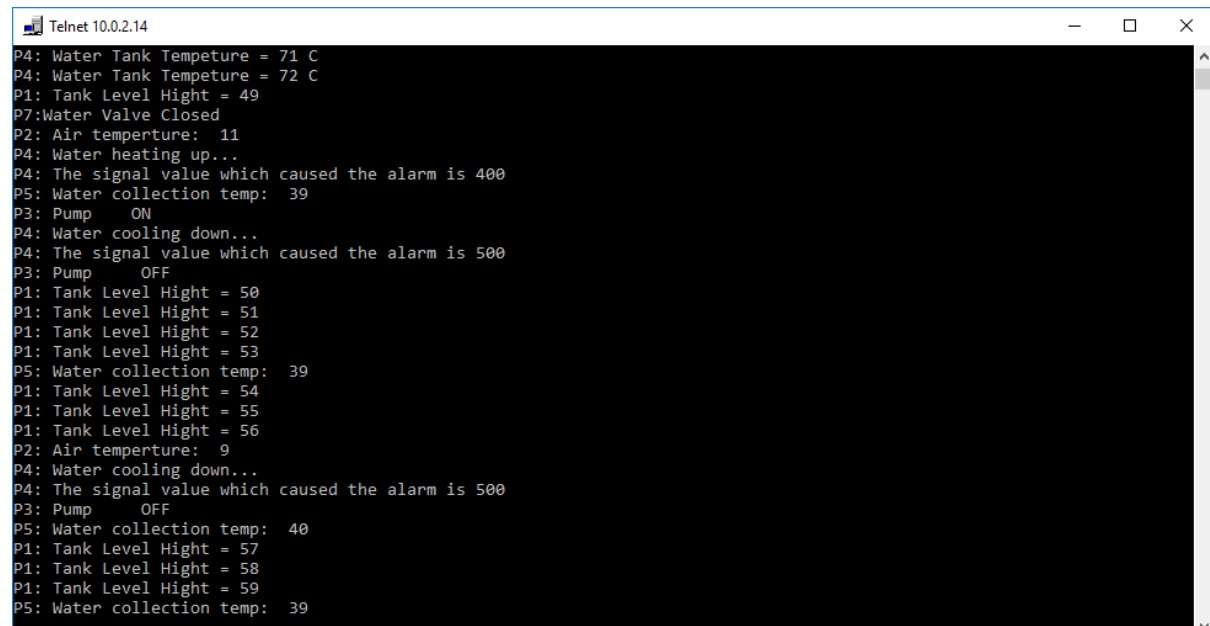
```
#define MESS_SZee 30
struct waterDataConstants
{
    int MessageNumbersConstants;
    int tankMaxHeight;
    int pumpSpeedMax;
};

#define MEMORY_WATER_CONSTANTS_SIZE sizeof(struct waterDataConstants)
```

Assignment 1.4 Running



```
Telnet 10.0.2.14
November 29, 2016 Tuesday 4:57:24 pm
[1]$ startup
Successfully created memory module
Successfully created memory module MEMORY_WATER
Successfully created memory module MEMORY_WATER_CONSTANTS
P4: Water Tank Tempeture = 66 C
P1: Tank Level Hight = 54
P4: Water Tank Tempeture = 67 C
P1: Tank Level Hight = 53
P4: Water Tank Tempeture = 68 C
P1: Tank Level Hight = 52
P4: Water Tank Tempeture = 69 C
P5: Water collection temp: 40
P1: Tank Level Hight = 51
P4: Water Tank Tempeture = 70 C
P1: Tank Level Hight = 50
P4: Water Tank Tempeture = 71 C
P4: Water Tank Tempeture = 72 C
P1: Tank Level Hight = 49
P7:Water Valve Closed
P2: Air temperture: 11
P4: Water heating up...
P4: The signal value which caused the alarm is 400
P5: Water collection temp: 39
P3: Pump ON
P4: Water cooling down...
P4: The signal value which caused the alarm is 500
P3: Pump OFF
P1: Tank Level Hight = 50
```



```
Telnet 10.0.2.14
P4: Water Tank Tempeture = 71 C
P4: Water Tank Tempeture = 72 C
P1: Tank Level Hight = 49
P7:Water Valve Closed
P2: Air temperture: 11
P4: Water heating up...
P4: The signal value which caused the alarm is 400
P5: Water collection temp: 39
P3: Pump ON
P4: Water cooling down...
P4: The signal value which caused the alarm is 500
P3: Pump OFF
P1: Tank Level Hight = 50
P1: Tank Level Hight = 51
P1: Tank Level Hight = 52
P1: Tank Level Hight = 53
P5: Water collection temp: 39
P1: Tank Level Hight = 54
P1: Tank Level Hight = 55
P1: Tank Level Hight = 56
P2: Air temperture: 9
P4: Water cooling down...
P4: The signal value which caused the alarm is 500
P3: Pump OFF
P5: Water collection temp: 40
P1: Tank Level Hight = 57
P1: Tank Level Hight = 58
P1: Tank Level Hight = 59
P5: Water collection temp: 39
```

```
Telnet 10.0.2.14
P3: Pump OFF
P1: Tank Level Hight = 50
P1: Tank Level Hight = 51
P1: Tank Level Hight = 52
P1: Tank Level Hight = 53
P5: Water collection temp: 39
P1: Tank Level Hight = 54
P1: Tank Level Hight = 55
P1: Tank Level Hight = 56
P2: Air temperture: 9
P4: Water cooling down...
P4: The signal value which caused the alarm is 500
P3: Pump OFF
P5: Water collection temp: 40
P1: Tank Level Hight = 57
P1: Tank Level Hight = 58
P1: Tank Level Hight = 59
P5: Water collection temp: 39
P1: Tank Level Hight = 60
P1: Tank Level Hight = 61
P1: Tank Level Hight = 62
P1: Tank Level Hight = 63
P2: Air temperture: 9
P4: Water cooling down...
P4: The signal value which caused the alarm is 500
P3: Pump OFF
P6: Solar Panel Tilt: 22 degree angle
P5: Water collection temp: 38
P1: Tank Level Hight = 64
```

```
Telnet 10.0.2.14
P5: Water collection temp: 40
P1: Tank Level Hight = 57
P1: Tank Level Hight = 58
P1: Tank Level Hight = 59
P5: Water collection temp: 39
P1: Tank Level Hight = 60
P1: Tank Level Hight = 61
P1: Tank Level Hight = 62
P1: Tank Level Hight = 63
P2: Air temperture: 9
P4: Water cooling down...
P4: The signal value which caused the alarm is 500
P3: Pump OFF
P6: Solar Panel Tilt: 22 degree angle
P5: Water collection temp: 38
P1: Tank Level Hight = 64
P1: Tank Level Hight = 65
P1: Tank Level Hight = 66
P5: Water collection temp: 39
P1: Tank Level Hight = 67
P1: Tank Level Hight = 68
P1: Tank Level Hight = 69
P2: Air temperture: 10
P4: Water heating up...
P4: The signal value which caused the alarm is 400
P3: Pump ON
P5: Water collection temp: 40
P1: Tank Level Hight = 70
P4: Water Tank Temperture = 73 C
```

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