Joel Satkauskas: R00116315

I, Joel Satkauskas, declare that the following solution for Problem 1.4 is entirely my own work.

Real Time Systems

Problem 1.4

Table of Contents

**Specification**2

**System3**

Diagram3

Specification3

**PIDStruct.h4**

**WaterStruct.h5**

**Stable.h7**

**P1.c8**

**P2.c**1**4**

**P3.c22**

**P4.c30**

**P5.c37**

**P6.c44**

**P7.c52**

**System Testing57**

Explanation64

**SPECIFICATION:**

**PROBLEM 1.4 [Use three data modules and signals required] (42)**

*Write a startup program to fork 7 processes. Three data modules are required; the first for an array called PID is used to store the process ID of each of the 7 forked processes. The two other (R/W) data modules store data relating to the management of your system. A struct relating to your water system with 7 fields is required.*

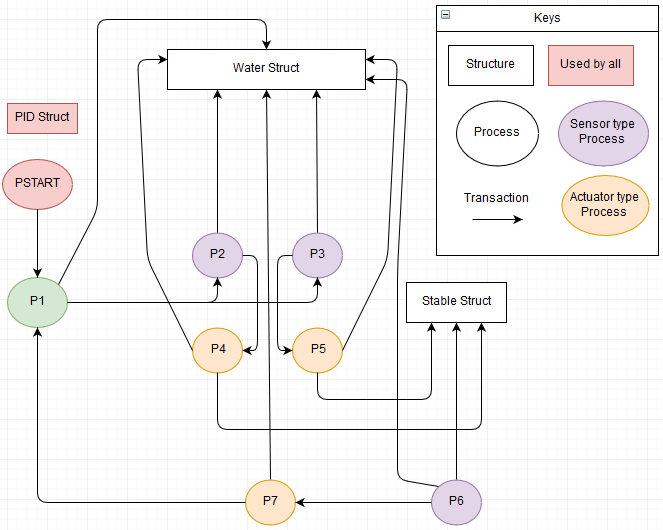
*Use your own ideas to illustrate your understanding of data modules, event and signals.*

*[Note: You must clearly specify your system; specify how each operation shows your understanding of a problem. Diagram(s) required.]*

*[Note: You are required to explain the logic of your solution in practical class.]*

**System**

**Diagram:**



**Specification:**

This system simulates a water treatment system were water would be entered into a tank, sensors would measure the waters properties, actuators would change the waters properties and when the water is in a suitable state, will pump the water somewhere else and repeat the process.

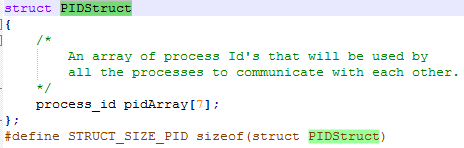
**PIDStruct.h**

**Status:** Finished. Working 100%.

**Spec:**

*Three data modules are required; the first for an array called PID is used to store the process ID of each of the 7 forked processes.*

**Code Snippet:**



**Source Code:**

struct PIDStruct

{

/\*

An array of process Id's that will be used by

all the processes to communicate with each other.

\*/

process\_id pidArray[7];

};

#define STRUCT\_SIZE\_PID sizeof(struct PIDStruct)

**Explanation:**

* This structure stores an array of process ids that will be used by the rest of the processes to communicate with each other.

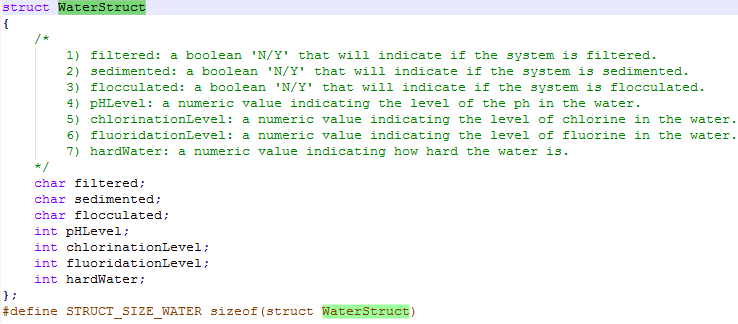
**WaterStruct.h**

**Status:** Finished. Working 100%.

**Spec:**

*The two other (R/W) data modules store data relating to the management of your system. A struct relating to your water system with 7 fields is required.*

**Code Snippet:**



**Source Code:**

struct WaterStruct

{

/\*

1) filtered: a boolean 'N/Y' that will indicate if

the system is filtered.

2) sedimented: a boolean 'N/Y' that will indicate if

the system is sedimented.

3) flocculated: a boolean 'N/Y' that will indicate

if the system is flocculated.

4) pHLevel: a numeric value indicating the level of

the ph in the water.

5) chlorinationLevel: a numeric value indicating the

level of chlorine in the water.

6) fluoridationLevel: a numeric value indicating the

level of fluorine in the water.

7) hardWater: a numeric value indicating how hard

the water is.

\*/

char filtered;

char sedimented;

char flocculated;

int pHLevel;

int chlorinationLevel;

int fluoridationLevel;

int hardWater;

};

#define STRUCT\_SIZE\_WATER sizeof(struct WaterStruct)

**Explanation:**

* This structure stores 7 fields that relate to the system.
* 1) filtered:

A char data type that is used as a Boolean, this value indicates whether the water is filtered or not, ‘Y/N’.

* 2) sedimented:

A char data type that is used as a Boolean, this value indicates whether the water is sedimented or not, ‘Y/N’.

* 3) flocculated:

A char data type that is used as a Boolean, this value indicates whether the water is flocculated or not, ‘Y/N’.

* 4) pHLevel:

An integer value to represent the level of the PH in the water.

* 5) chlorinationLevel:

An integer value to represent the level of chlorine in the water.

* 6) fluoridationLevel:

An integer value to represent the level of fluorine in the water.

* 7) hardWater:

An integer value to represent the hardness of the water.

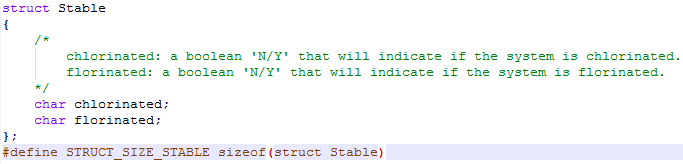
**Stable.h**

**Status:** Finished. Working 100%.

**Spec:**

*The two other (R/W) data modules store data relating to the management of your system.*

**Code Snippet:**



**Source Code:**

struct Stable

{

/\*

chlorinated: a boolean 'N/Y' that will indicate if the system is chlorinated.

florinated: a boolean 'N/Y' that will indicate if the system is florinated.

\*/

char chlorinated;

char florinated;

};

#define STRUCT\_SIZE\_STABLE sizeof(struct Stable)

**Explanation:**

* This structure stores 2 char data types that are used to represent Booleans.
* 1) chlorinated:

A value that represents whether the system has chlorinated the water. ‘Y/N’

* 2) florinated:

A value that represents whether the system has fluorinated the water. ‘Y/N’

**P1.c**

**Status:** Finished. Working 100%.

**Source Code:**

#include <stdio.h>

#include <signal.h>

#include <types.h>

#include <modes.h>

#include <errno.h>

#include <cglob.h>

#include <stdlib.h>

#include "PIDStruct.h"

#include "WaterStruct.h"

#define PID\_ARRAY\_NAME "PID"

#define WATER\_STRUCT\_NAME "WaterStruct"

/\*

MIN\_N is the minimum number for a random number to be generated

\*/

#define MIN\_N 1

/\*

MAX\_N is the maximum number for a random number to be generated

\*/

#define MAX\_N 100

/\*

WATER\_WAIT\_TIME is the amount of milliseconds to wait for the water to enter the system.

\*/

#define WATER\_WAIT\_TIME 5000

u\_int32 SleepValue;

mh\_com mod\_head;

char \*ptrPIDName;

char \*ptrWaterName;

struct PIDStruct \*PID;

struct WaterStruct \*waterStruct;

signal\_code DummySignal;

signal\_code dummy\_sig;

event\_id ev\_id;

u\_int32 value;

int linked = 0;

u\_int32 milSecs = 3000;

u\_int16 attr\_rev = (MA\_REENT << 8);

u\_int16 type\_lang = (MT\_DATA << 8);

u\_int16 mem\_size\_pid = STRUCT\_SIZE\_PID;

u\_int16 mem\_size\_water = STRUCT\_SIZE\_WATER;

sig\_handler(signal\_code sig)

{

switch(sig)

{

case 3:

printf("\nP1: Signal to shut down\n");

\_os\_exit(0);

break;

}

\_os\_rte();

}

/\*

link():

A method to link to all the recources needed by the

process.

Once linked, it will set 'linked' to 1 to indicate that everything is already linked

so it wont try to link again.

\*/

link()

{

ptrPIDName = PID\_ARRAY\_NAME;

ptrWaterName = WATER\_STRUCT\_NAME;

errno = \_os\_link(&ptrPIDName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&PID,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P1: %d: Couldnt Link to data pid

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrWaterName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&waterStruct,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P1: %d: Couldnt Link to water data

module\n", \_procid);

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(WATER\_STRUCT\_NAME,

&ev\_id)) != 0)

{

printf("P1: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

linked = 1;

}

/\*

nextInput():

A method that will access the water structure by using

events and assign the

water structure random values to simulate a new entry of

water.

\*/

nextInput()

{

signal\_code signal;

int newPhLevel = getRandNum();

int newChLevel = getRandNum();

int newFlLevel = getRandNum();

int newHW = getRandNum();

printf("P1: Pumping next 1000L of water into system\n");

sleepFor(WATER\_WAIT\_TIME);

printf("----------------------------\n");

printf("PHLevel is %d \n",newPhLevel);

printf("Chlorination Level is %d \n",newChLevel);

printf("Fluorination Level is %d \n",newFlLevel);

printf("HardWater level is %d \n",newHW);

printf("----------------------------\n");

if((errno = \_os\_ev\_wait(ev\_id,

&value,

&signal,

1,

1)) != 0)

{

printf("P1: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterStruct->pHLevel = newPhLevel;

waterStruct->chlorinationLevel = newChLevel;

waterStruct->fluoridationLevel = newFlLevel;

waterStruct->hardWater = newHW;

waterStruct->filtered = 'N';

waterStruct->sedimented = 'N';

waterStruct->flocculated = 'N';

if((errno = \_os\_ev\_signal(ev\_id,

&value,

0)) != 0)

{

printf("P1: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

}

/\*

getRandNum():

A method that will get a random number between the global

variables of MIN\_N and MAX\_N

Note: this method does not return truly random values but

for this simulation will suffice

\*/

int getRandNum()

{

return (rand() % (MAX\_N - MIN\_N + 1) + MIN\_N);

}

/\*

sleepFor(int milSecs):

A method that will take in an int value to sleep for. No

signal will interupt its sleep.

milSecs: amount of milliseconds to sleep for.

\*/

sleepFor(int milSecs)

{

signal\_code dummySig;

int sleepFor = milSecs;

while(sleepFor)

{

\_os\_sleep(&sleepFor, &dummySig);

}

}

main()

{

error\_code err;

u\_int32 milSecs;

signal\_code ReceivedSignal;

if((err = \_os\_intercept(sig\_handler, \_glob\_data)) != 0)

{

printf("P1: Failed to attach signal handler\n");

exit(err);

}

milSecs = 0;

while(1)

{

\_os\_sleep(&milSecs, &ReceivedSignal);

if(!linked)

link();

if(ReceivedSignal == 356)

{

/\*

call next input to make new struct of random

values.

call next processes to start using it

\*/

nextInput();

printf("P1: Water ready. Calling

sensors.\n\n\n");

if(err = (\_os\_send(PID->pidArray[1], 357))!=0)

{

printf("P1: Failed to send Start to 2\n");

}

if(err = (\_os\_send(PID->pidArray[2], 357))!= 0)

{

printf("P1: Failed to send Start to

P3\n");

}

}

}

}

**Explanation:**

* P1 is to simulate a process entering water into the system and then letting the sensors know they can start working, once it is done.
* Once P1 is set up and receives a signal of 356, it will wake up and check if it is linked to all the modules needed by it.

It does this every time it wakes up by a signal regardless of the signal.

If it is unlinked, it will call the link() method that will link to the water and PID structures as well as the water event. It will change the Boolean to indicate that it has linked and so wont link again for the duration of its life time.

* If the signal called is 356 it will call the method nextInput() that will assign random values to the water structure using events.
* Once this is done it will send a signal to P2 and P3, turning them on so that they can continue with the system.

**P2.c**

**Status:** Finished. Working 100%.

**Source Code:**

#include <stdio.h>

#include <signal.h>

#include <types.h>

#include <modes.h>

#include <errno.h>

#include <cglob.h>

#include <stdlib.h>

#include "PIDStruct.h"

#include "WaterStruct.h"

#define PID\_ARRAY\_NAME "PID"

#define WATER\_STRUCT\_NAME "WaterStruct"

/\*

SENSOR\_FREQUENCY: the amount of milliseconds to sleep until the alarm wakes up the process again.

\*/

#define SENSOR\_FREQUENCY 1500

/\*

SENSOR\_Value: the value for which the process will try to either be above or below

\*/

#define SENSOR\_Value 70

/\*

SENSOR\_DIRECTION: A boolean value to indicate wheather it is desired to have the water value above or below the SENSOR\_Value.

\*/

#define SENSOR\_DIRECTION 1

u\_int32 SleepValue;

mh\_com mod\_head;

char \*ptrPIDName;

char \*ptrWaterName;

struct PIDStruct \*PID;

struct WaterStruct \*waterStruct;

signal\_code DummySignal;

signal\_code dummy\_sig;

event\_id ev\_id;

u\_int32 value;

/\*

on: a boolean to indicate wheather the sensor is on or

off, wheather to read the water structure and perform operations or to ignore the

alarm.

\*/

int on = 0;

int linked = 0;

u\_int32 milSecs = 3000;

u\_int16 attr\_rev = (MA\_REENT << 8);

u\_int16 type\_lang = (MT\_DATA << 8);

u\_int16 mem\_size\_pid = STRUCT\_SIZE\_PID;

u\_int16 mem\_size\_water = STRUCT\_SIZE\_WATER;

sig\_handler(signal\_code sig)

{

switch(sig)

{

case 3:

printf("\nP2: Signal to shut down\n");

\_os\_exit(0);

break;

case 357:

printf("P2: Received message to turn on\n");

on = 1;

}

\_os\_rte();

}

/\*

link():

A method to link to all the recources needed by the

process.

Once linked, it will set 'linked' to 1 to indicate that everything is already linked

so it wont try to link again.

\*/

link()

{

ptrPIDName = PID\_ARRAY\_NAME;

ptrWaterName = WATER\_STRUCT\_NAME;

errno = \_os\_link(&ptrPIDName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&PID,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P2: %d: Couldnt Link to data pid

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrWaterName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&waterStruct,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P2: %d: Couldnt Link to water data

module\n", \_procid);

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(WATER\_STRUCT\_NAME,

&ev\_id)) != 0)

{

printf("P2: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

linked = 1;

}

/\*

sleepFor(int milSecs):

A method that will take in an int value to sleep for. No

signal will interupt its sleep.

milSecs: amount of milliseconds to sleep for.

\*/

sleepFor(int milSecs)

{

signal\_code dummySig;

int sleepFor = milSecs;

while(sleepFor)

{

\_os\_sleep(&sleepFor, &dummySig);

}

}

/\*

readWater():

A method that will retrieve the value fluoridationLevel

from the water structure by using events.

\*/

int readWater()

{

signal\_code signal;

int waterValue;

if((errno = \_os\_ev\_wait(ev\_id,

&value,

&signal,

1,

1)) != 0)

{

printf("P2: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterValue = waterStruct->fluoridationLevel;

if((errno = \_os\_ev\_signal(ev\_id,

&value,

0)) != 0)

{

printf("P2: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

return waterValue;

}

main()

{

error\_code err;

u\_int32 milSecs;

signal\_code ReceivedSignal;

alarm\_id MyAlarm;

signal\_code WakeupSignal;

u\_int32 TimeToDelay;

int waterValue;

if((err = \_os\_intercept(sig\_handler, \_glob\_data)) != 0)

{

printf("P2: Failed to attach signal handler\n");

exit(err);

}

/\*

WakeupSignal: the signal for the cyclic alarm to use when calling.

\*/

WakeupSignal = 356;

TimeToDelay = SENSOR\_FREQUENCY;

if((errno = \_os\_alarm\_cycle(&MyAlarm,

WakeupSignal,

TimeToDelay)) !=0)

{

printf("P2: error creating alarm\n");

exit(errno);

}

milSecs = 0;

while(1)

{

\_os\_sleep(&milSecs, &ReceivedSignal);

if(!linked)

link();

if(ReceivedSignal == 356)

{

if(on)

{

waterValue = readWater();

printf("P2: Fluoridation Level:

%d\n",waterValue);

/\*

if we want it to be above the value

\*/

if(SENSOR\_DIRECTION)

{

if(waterValue < SENSOR\_Value)

{

/\*

send message to increase

fluoridationLevel

\*/

printf("P2: Sending message to

Increase\n\n");

if(err =

(\_os\_send(PID>pidArray[3], 355))

!= 0)

{

printf("P2: Failed to send

Increase to P4\n");

}

}

else

{

/\*

level satisfactory, send to finish and

turn off.

\*/

printf("P2: Fluoridation Level

satisfactory\n");

printf("P2: Sending finish to

P4\n\n");

if(err =

(\_os\_send(PID->pidArray[3], 357))

!= 0)

{

printf("P2: Failed to send

Finish to P4\n");

}

else

{

on = 0;

}

}

}

else

{

/\*

else we want it to be below the value

\*/

if(waterValue >= SENSOR\_Value)

{

/\*

send message to decrease

fluoridationLevel

\*/

printf("P2: Sending message to

Decrease\n\n");

if(err =

(\_os\_send(PID->pidArray[3], 356))

!= 0)

{

printf("P2: Failed to send

Decrease to P4\n");

}

}

else

{

/\*

level satisfactory,

send to

finish and turn off.

\*/

printf("P2: Fluoridation

Level satisfactory\n");

printf("P2: Sending finish

to P4\n\n");

if(err =

(\_os\_send(PID->pidArray[3],

357)) != 0)

{

printf("P2: Failed to

send Finish to P4\n");

}

else

{

on = 0;

}

}

}

}

}

}

}

**Explanation:**

* P2 is to simulate a sensor that will always be reading the values of the water structure and deciding what to do base on that value.
* P2 has a cyclic alarm that will wake it up every 1.5 seconds. Every time it wakes up, it checks to see if it is supposed to be on. If not it ignores the signal and goes back to sleep, if it is it will continue its procedure.
* If it wakes up and it’s ‘on’ Boolean is true, then it will proceed to read the water value using events and decide whether to send a message to P4 to increase, decrease or be finished with the water structure.

It does this based on 2 global variables, SENSOR\_Value and SENSOR\_DIRECTION.

SENSOR\_Value is an integer value that will indicate whether the process wants to be above or below this value for the ‘fluoridationLevel’ in the water structure.

SENSOR\_DIRECTION is a Boolean value that will indicate whether the system wants the water structures ‘fluoridationLevel’ value above or below SENSOR\_Value.

E.g. An instance were:

SENSOR\_Value = 40 and SENSOR\_DIRECTION = 1

Means the system wants the ‘fluoridationLevel’ value to be above 40.

An instance were:

SENSOR\_Value = 80 and SENSOR\_DIRECTION = 0

Means the system wants the ‘fluoridationLevel’ value to be below 80.

* Based on this logic, it will send a message to P4 to:
  + Increase the level or
  + Decrease the level or
  + Finish.

Once it calls finish it will turn its self off.

**P3.c**

**Status:** Finished. Working 100%.

**Source Code:**

#include <stdio.h>

#include <signal.h>

#include <types.h>

#include <modes.h>

#include <errno.h>

#include <cglob.h>

#include <stdlib.h>

#include "PIDStruct.h"

#include "WaterStruct.h"

#define PID\_ARRAY\_NAME "PID"

#define WATER\_STRUCT\_NAME "WaterStruct"

/\*

SENSOR\_FREQUENCY: the amount of milliseconds to sleep

until the alarm wakes up the process again.

\*/

#define SENSOR\_FREQUENCY 1500

/\*

SENSOR\_Value: the value for which the process will try to either be above or below

\*/

#define SENSOR\_Value 40

/\*

SENSOR\_DIRECTION: A boolean value to indicate wheather it is desired to

have the water value

above or below the SENSOR\_Value.

1 - above SENSOR\_Value

0 - below SENSOR\_Value

\*/

#define SENSOR\_DIRECTION 0

u\_int32 SleepValue;

mh\_com mod\_head;

char \*ptrPIDName;

char \*ptrWaterName;

struct PIDStruct \*PID;

struct WaterStruct \*waterStruct;

signal\_code DummySignal;

signal\_code dummy\_sig;

event\_id ev\_id;

u\_int32 value;

/\*

on: a boolean to indicate wheather the sensor is on or off, wheather to read the water structure and perform operations or to ignore the alarm.

\*/

int on = 0;

int linked = 0;

u\_int32 milSecs = 3000;

u\_int16 attr\_rev = (MA\_REENT << 8);

u\_int16 type\_lang = (MT\_DATA << 8);

u\_int16 mem\_size\_pid = STRUCT\_SIZE\_PID;

u\_int16 mem\_size\_water = STRUCT\_SIZE\_WATER;

sig\_handler(signal\_code sig)

{

switch(sig)

{

case 3:

printf("\nP3: Signal to shut down\n");

\_os\_exit(0);

break;

case 357:

printf("P3: Received message to turn on\n");

on = 1;

}

\_os\_rte();

}

/\*

link():

A method to link to all the recources needed by the

process.

Once linked, it will set 'linked' to 1 to indicate that everything is already linked so it wont try to link again.

\*/

link()

{

ptrPIDName = PID\_ARRAY\_NAME;

ptrWaterName = WATER\_STRUCT\_NAME;

errno = \_os\_link(&ptrPIDName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&PID,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P3: %d: Couldnt Link to data pid

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrWaterName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&waterStruct,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P3: %d: Couldnt Link to water data

module\n", \_procid);

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(WATER\_STRUCT\_NAME,

&ev\_id)) != 0)

{

printf("P3: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

linked = 1;

}

/\*

readWater():

A method that will retrieve the value chlorinationLevel

from the water structure by using events.

\*/

int readWater()

{

signal\_code signal;

int waterValue;

if((errno = \_os\_ev\_wait(ev\_id,

&value,

&signal,

1,

1)) != 0)

{

printf("P3: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterValue = waterStruct->chlorinationLevel;

if((errno = \_os\_ev\_signal(ev\_id,

&value,

0)) != 0)

{

printf("P3: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

return waterValue;

}

/\*

sleepFor(int milSecs):

A method that will take in an int value to sleep for. No

signal will interupt its sleep.

milSecs: amount of milliseconds to sleep for.

\*/

sleepFor(int milSecs)

{

signal\_code dummySig;

int sleepFor = milSecs;

while(sleepFor)

{

\_os\_sleep(&sleepFor, &dummySig);

}

}

main()

{

error\_code err;

u\_int32 milSecs;

signal\_code ReceivedSignal;

alarm\_id MyAlarm;

signal\_code WakeupSignal;

u\_int32 TimeToDelay;

int waterValue;

if((err = \_os\_intercept(sig\_handler, \_glob\_data)) != 0)

{

printf("P3: Failed to attach signal handler\n");

exit(err);

}

/\*

WakeupSignal: the signal for the cyclic alarm to use

when calling.

\*/

WakeupSignal = 356;

TimeToDelay = SENSOR\_FREQUENCY;

if((errno = \_os\_alarm\_cycle(&MyAlarm,

WakeupSignal,

TimeToDelay)) !=0)

{

printf("P3: error creating alarm\n");

exit(errno);

}

milSecs = 0;

while(1)

{

\_os\_sleep(&milSecs, &ReceivedSignal);

if(!linked)

link();

if(ReceivedSignal == 356)

{

if(on)

{

waterValue = readWater();

printf("P3: Chlorination Level:

%d\n",waterValue);

/\*

if we want it to be above the value

\*/

if(SENSOR\_DIRECTION)

{

if(waterValue < SENSOR\_Value)

{

/\*

send message to increase

chlorinationLevel

\*/

printf("P3: Sending message to

Increase\n\n");

if(err =

(\_os\_send(PID->pidArray[3], 355))

!= 0)

{

printf("P3: Failed to send

Increase to P5\n");

}

}

else

{

/\*

level satisfactory, send to finish and turn off.

\*/

printf("P3: Chlorination Level

satisfactory\n");

printf("P3: Sending finish to

P5\n\n");

if(err =

(\_os\_send(PID->pidArray[3], 357))

!= 0)

{

printf("P3: Failed to send

Finish to P5\n");

}

else

{

on = 0;

}

}

}

else

{

/\*

else we want it to be below the

value

\*/

if(waterValue >= SENSOR\_Value)

{

/\*

send message to decrease

chlorinationLevel

\*/

printf("P3: Sending message to

Decrease\n\n");

if(err =

(\_os\_send(PID->pidArray[4], 356))

!= 0)

{

printf("P3: Failed to send

Decrease to P5\n");

}

}

else

{

/\*

level satisfactory,

send to

finish and turn off.

\*/

printf("P3: Chlorination

Level satisfactory\n");

printf("P3: Sending finish

to P5\n\n");

if(err =

(\_os\_send(PID->pidArray[4],

357)) != 0)

{

printf("P3: Failed to

send Finish to P5\n");

}

else

{

on = 0;

}

}

}

}

}

}

}

**Explanation:**

* P3 is to simulate a sensor that will always be reading the values of the water structure and deciding what to do base on that value. It is essentially the same as P2 except it will be making its decisions on the ‘chlorinationLevel’ value and sending its messages to P5.
* P3 has a cyclic alarm that will wake it up every 1.5 seconds. Every time it wakes up, it checks to see if it is supposed to be on. If not it ignores the signal and goes back to sleep, if it is it will continue its procedure.
* If it wakes up and it’s ‘on’ Boolean is true, then it will proceed to read the water value using events and decide whether to send a message to P5 to increase, decrease or be finished with the water structure.

It does this based on 2 global variables, SENSOR\_Value and SENSOR\_DIRECTION.

SENSOR\_Value is an integer value that will indicate whether the process wants to be above or below this value for the ‘chlorinationLevel’ in the water structure.

SENSOR\_DIRECTION is a Boolean value that will indicate whether the system wants the water structures ‘chlorinationLevel’ value above or below SENSOR\_Value.

E.g. An instance were:

SENSOR\_Value = 40 and SENSOR\_DIRECTION = 1

Means the system wants the ‘chlorinationLevel’ value to be above 40.

An instance were:

SENSOR\_Value = 80 and SENSOR\_DIRECTION = 0

Means the system wants the ‘chlorinationLevel’ value to be below 80.

* Based on this logic, it will send a message to P5 to:
  + Increase the level or
  + Decrease the level or
  + Finish.

Once it calls finish it will turn its self off.

**P4.c**

**Status:** Finished. Working 100%.

**Source Code:**

#include <stdio.h>

#include <signal.h>

#include <types.h>

#include <modes.h>

#include <errno.h>

#include <cglob.h>

#include <stdlib.h>

#include "PIDStruct.h"

#include "WaterStruct.h"

#include "Stable.h"

#define PID\_ARRAY\_NAME "PID"

#define WATER\_STRUCT\_NAME "WaterStruct"

#define STABLE\_STRUCT\_NAME "StableStruct"

/\*

CHANGE\_BY: the value of which to change the main value of the water structure.

\*/

#define CHANGE\_BY 5

/\*

CHANGE\_BYPRODUCT: the value of which to change the by product value of the water structure,

simulates the change of anoither aspect when changing one value.

\*/

#define CHANGE\_BYPRODUCT 2

u\_int32 SleepwaterValue;

mh\_com mod\_head;

char \*ptrPIDName;

char \*ptrWaterName;

char \*ptrStableName;

struct PIDStruct \*PID;

struct WaterStruct \*waterStruct;

struct Stable \*stable;

signal\_code DummySignal;

signal\_code dummy\_sig;

event\_id water\_ev\_id;

event\_id stable\_ev\_id;

u\_int32 waterValue;

u\_int32 stableValue;

int linked = 0;

u\_int32 milSecs = 3000;

u\_int16 attr\_rev = (MA\_REENT << 8);

u\_int16 type\_lang = (MT\_DATA << 8);

u\_int16 mem\_size\_pid = STRUCT\_SIZE\_PID;

u\_int16 mem\_size\_water = STRUCT\_SIZE\_WATER;

sig\_handler(signal\_code sig)

{

switch(sig)

{

case 3:

printf("\nP4: Signal to shut down\n");

\_os\_exit(0);

break;

}

\_os\_rte();

}

/\*

link():

A method to link to all the recources needed by the

process.

Once linked, it will set 'linked' to 1 to indicate that

everything is already linked so it wont try to link

again.

\*/

link()

{

ptrPIDName = PID\_ARRAY\_NAME;

ptrWaterName = WATER\_STRUCT\_NAME;

ptrStableName = STABLE\_STRUCT\_NAME;

errno = \_os\_link(&ptrPIDName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&PID,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P4: %d: Couldnt Link to data pid

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrWaterName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&waterStruct,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P4: %d: Couldnt Link to water data

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrStableName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&stable,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P4: %d: Couldnt Link to Stable data

module\n", \_procid);

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(WATER\_STRUCT\_NAME,

&water\_ev\_id)) != 0)

{

printf("P4: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(STABLE\_STRUCT\_NAME,

&stable\_ev\_id)) != 0)

{

printf("P4: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

linked = 1;

}

/\*

sleepFor(int milSecs):

A method that will take in an int value to sleep for. No

signal will interupt its sleep.

milSecs: amount of milliseconds to sleep for.

\*/

sleepFor(int milSecs)

{

signal\_code dummySig;

int sleepFor = milSecs;

while(sleepFor)

{

\_os\_sleep(&sleepFor, &dummySig);

}

}

/\*

IncreaseWater():

A method that will increase values in the water structure

by using events.

\*/

IncreaseWater()

{

signal\_code signal;

if((errno = \_os\_ev\_wait(water\_ev\_id,

&waterValue,

&signal,

1,

1)) != 0)

{

printf("P4: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterStruct->fluoridationLevel += CHANGE\_BY;

waterStruct->hardWater += CHANGE\_BYPRODUCT;

if((errno = \_os\_ev\_signal(water\_ev\_id,

&waterValue,

0)) != 0)

{

printf("P4: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

}

/\*

DecreaseWater():

A method that will decrease values in the water structure

by using events.

\*/

DecreaseWater()

{

signal\_code signal;

if((errno = \_os\_ev\_wait(water\_ev\_id,

&waterValue,

&signal,

1,

1)) != 0)

{

printf("P4: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterStruct->fluoridationLevel -= CHANGE\_BY;

waterStruct->hardWater -= CHANGE\_BYPRODUCT;

if((errno = \_os\_ev\_signal(water\_ev\_id,

&waterValue,

0)) != 0)

{

printf("P4: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

}

/\*

waterFinished():

A method that will change a boolean on the stable

structure using events

\*/

waterFinished()

{

signal\_code signal;

if((errno = \_os\_ev\_wait(stable\_ev\_id,

&stableValue,

&signal,

1,

1)) != 0)

{

printf("P4: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

stable->florinated = 'Y';

if((errno = \_os\_ev\_signal(stable\_ev\_id,

&stableValue,

0)) != 0)

{

printf("P4: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

}

main()

{

error\_code err;

u\_int32 milSecs;

signal\_code ReceivedSignal;

if((err = \_os\_intercept(sig\_handler, \_glob\_data)) != 0)

{

printf("P4: Failed to attach signal handler\n");

exit(err);

}

milSecs = 0;

while(1)

{

\_os\_sleep(&milSecs, &ReceivedSignal);

if(!linked)

link();

if(ReceivedSignal == 355)

{

/\*

Increase fluoridationLevel

\*/

printf("P4: Increasing fluoridationLevel by:

%d\n",CHANGE\_BY);

IncreaseWater();

}

else if(ReceivedSignal == 356)

{

/\*

decrease fluoridationLevel

\*/

printf("P4: Decreasing fluoridationLevel by: %d

\n",CHANGE\_BY);

DecreaseWater();

}

else if(ReceivedSignal == 357)

{

/\*

Finish

\*/

printf("P4: fluoridationLevel Stabilized,

Sending to Struct \n\n\n");

waterFinished();

}

}

}

**Explanation:**

* P4 is to simulate an actuator that will release chemicals into the water to change its values.
* P4 will sleep until it receives messages from P2 to act upon. These messages could be:
* 355:

This message will make P4 increase the value on the water structure.

* 356:

This message will make P4 decrease the value on the water structure.

* 357:

This message will make P4 change the ‘florinated’ value on the stable structure to a ‘Y’ indicating that the water has been fluorinated.

**P5.c**

**Status:** Finished. Working 100%.

**Source Code:**

#include <stdio.h>

#include <signal.h>

#include <types.h>

#include <modes.h>

#include <errno.h>

#include <cglob.h>

#include <stdlib.h>

#include "PIDStruct.h"

#include "WaterStruct.h"

#include "Stable.h"

#define PID\_ARRAY\_NAME "PID"

#define WATER\_STRUCT\_NAME "WaterStruct"

#define STABLE\_STRUCT\_NAME "StableStruct"

/\*

CHANGE\_BY: the value of which to change the main value of

the water structure.

\*/

#define CHANGE\_BY 3

/\*

CHANGE\_BYPRODUCT: the value of which to change the by

product value of the water structure,

simulates the change of anoither aspect when changing one

value.

\*/

#define CHANGE\_BYPRODUCT 1

u\_int32 SleepValue;

mh\_com mod\_head;

char \*ptrPIDName;

char \*ptrWaterName;

char \*ptrStableName;

struct PIDStruct \*PID;

struct WaterStruct \*waterStruct;

struct Stable \*stable;

signal\_code DummySignal;

signal\_code dummy\_sig;

event\_id water\_ev\_id;

event\_id stable\_ev\_id;

u\_int32 waterValue;

u\_int32 stableValue;

int linked = 0;

u\_int32 milSecs = 3000;

u\_int16 attr\_rev = (MA\_REENT << 8);

u\_int16 type\_lang = (MT\_DATA << 8);

u\_int16 mem\_size\_pid = STRUCT\_SIZE\_PID;

u\_int16 mem\_size\_water = STRUCT\_SIZE\_WATER;

sig\_handler(signal\_code sig)

{

switch(sig)

{

case 3:

printf("\nP5: Signal to shut down\n");

\_os\_exit(0);

break;

}

\_os\_rte();

}

/\*

link():

A method to link to all the recources needed by the

process.

Once linked, it will set 'linked' to 1 to indicate that everything is already linked so it wont try to link again.

\*/

link()

{

ptrPIDName = PID\_ARRAY\_NAME;

ptrWaterName = WATER\_STRUCT\_NAME;

ptrStableName = STABLE\_STRUCT\_NAME;

errno = \_os\_link(&ptrPIDName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&PID,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P5: %d: Couldnt Link to data pid

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrWaterName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&waterStruct,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P5: %d: Couldnt Link to water data

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrStableName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&stable,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P5: %d: Couldnt Link to Stable data

module\n", \_procid);

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(WATER\_STRUCT\_NAME,

&water\_ev\_id)) != 0)

{

printf("P5: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(STABLE\_STRUCT\_NAME,

&stable\_ev\_id)) != 0)

{

printf("P5: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

linked = 1;

}

/\*

sleepFor(int milSecs):

A method that will take in an int value to sleep for. No

signal will interupt its sleep.

milSecs: amount of milliseconds to sleep for.

\*/

sleepFor(int milSecs)

{

signal\_code dummySig;

int sleepFor = milSecs;

while(sleepFor)

{

\_os\_sleep(&sleepFor, &dummySig);

}

}

/\*

IncreaseWater():

A method that will increase values in the water structure

by using events.

\*/

IncreaseWater()

{

signal\_code signal;

if((errno = \_os\_ev\_wait(water\_ev\_id,

&waterValue,

&signal,

1,

1)) != 0)

{

printf("P5: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterStruct->chlorinationLevel += CHANGE\_BY;

waterStruct->pHLevel += CHANGE\_BYPRODUCT;

if((errno = \_os\_ev\_signal(water\_ev\_id,

&waterValue,

0)) != 0)

{

printf("P5: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

}

/\*

DecreaseWater():

A method that will decrease values in the water structure

by using events.

\*/

DecreaseWater()

{

signal\_code signal;

if((errno = \_os\_ev\_wait(water\_ev\_id,

&waterValue,

&signal,

1,

1)) != 0)

{

printf("P5: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterStruct->chlorinationLevel -= CHANGE\_BY;

waterStruct->pHLevel -= CHANGE\_BYPRODUCT;

if((errno = \_os\_ev\_signal(water\_ev\_id,

&waterValue,

0)) != 0)

{

printf("P5: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

}

/\*

waterFinished():

A method that will change a boolean on the stable

structure using events

\*/

waterFinished()

{

signal\_code signal;

if((errno = \_os\_ev\_wait(stable\_ev\_id,

&stableValue,

&signal,

1,

1)) != 0)

{

printf("P5: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

stable->chlorinated = 'Y';

if((errno = \_os\_ev\_signal(stable\_ev\_id,

&stableValue,

0)) != 0)

{

printf("P5: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

}

main()

{

error\_code err;

u\_int32 milSecs;

signal\_code ReceivedSignal;

if((err = \_os\_intercept(sig\_handler, \_glob\_data)) != 0)

{

printf("P5: Failed to attach signal handler\n");

exit(err);

}

milSecs = 0;

while(1)

{

\_os\_sleep(&milSecs, &ReceivedSignal);

if(!linked)

link();

if(ReceivedSignal == 355)

{

/\*

Increase chlorinationLevel

\*/

printf("P5: Increasing chlorinationLevel by: %d

\n",CHANGE\_BY);

IncreaseWater();

}

else if(ReceivedSignal == 356)

{

/\*

decrease chlorinationLevel

\*/

printf("P5: Decreasing chlorinationLevel by: %d

\n",CHANGE\_BY);

DecreaseWater();

}

else if(ReceivedSignal == 357)

{

/\*

Finish

\*/

printf("P5: chlorinationLevel Stabilized,

Sending to Struct \n\n\n");

waterFinished();

}

}

}

**Explanation:**

* P5 is to simulate an actuator that will release chemicals into the water to change its values.
* P5 will sleep until it receives messages from P3 to act upon. These messages could be:
* 355:

This message will make P5 increase the value on the water structure.

* 356:

This message will make P5 decrease the value on the water structure.

* 357:

This message will make P5 change the ‘chlorinated’ value on the stable structure to a ‘Y’ indicating that the water has been fluorinated.

**P6.c**

**Status:** Finished. Working 100%.

**Source Code:**

#include <stdio.h>

#include <signal.h>

#include <types.h>

#include <modes.h>

#include <errno.h>

#include <cglob.h>

#include <stdlib.h>

#include "PIDStruct.h"

#include "Stable.h"

#include "WaterStruct.h"

#define PID\_ARRAY\_NAME "PID"

#define STABLE\_STRUCT\_NAME "StableStruct"

#define WATER\_STRUCT\_NAME "WaterStruct"

/\*

SENSOR\_FREQUENCY: the amount of milliseconds to sleep

until the alarm wakes up the process again.

\*/

#define SENSOR\_FREQUENCY 3000

/\*

SLEEP\_FOR: the amount of milliseconds to sleep for before

sending message to P7.

\*/

#define SLEEP\_FOR 2000

u\_int32 SleepValue;

mh\_com mod\_head;

char \*ptrPIDName;

char \*ptrStableName;

char \*ptrWaterName;

struct PIDStruct \*PID;

struct Stable \*stable;

struct WaterStruct \*waterStruct;

signal\_code DummySignal;

signal\_code dummy\_sig;

event\_id stable\_ev\_id;

event\_id water\_ev\_id;

u\_int32 waterValue;

u\_int32 stableValue;

int linked = 0;

u\_int32 milSecs = 3000;

u\_int16 attr\_rev = (MA\_REENT << 8);

u\_int16 type\_lang = (MT\_DATA << 8);

u\_int16 mem\_size\_pid = STRUCT\_SIZE\_PID;

u\_int16 mem\_size\_water = STRUCT\_SIZE\_WATER;

sig\_handler(signal\_code sig)

{

switch(sig)

{

case 3:

printf("\nP6: Signal to shut down\n");

\_os\_exit(0);

break;

}

\_os\_rte();

}

/\*

link():

A method to link to all the recources needed by the

process.

Once linked, it will set 'linked' to 1 to indicate that

everything is already linked so it wont try to link again.

\*/

link()

{

ptrPIDName = PID\_ARRAY\_NAME;

ptrStableName = STABLE\_STRUCT\_NAME;

ptrWaterName = WATER\_STRUCT\_NAME;

errno = \_os\_link(&ptrPIDName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&PID,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P6: %d: Couldnt Link to data pid

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrWaterName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&waterStruct,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P6: %d: Couldnt Link to water data

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrStableName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&stable,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P6: %d: Couldnt Link to Stable data

module\n", \_procid);

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(STABLE\_STRUCT\_NAME,

&stable\_ev\_id)) != 0)

{

printf("P5: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(WATER\_STRUCT\_NAME,

&water\_ev\_id)) != 0)

{

printf("P5: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

linked = 1;

}

/\*

sleepFor(int milSecs):

A method that will take in an int value to sleep for. No

signal will interupt its sleep.

milSecs: amount of milliseconds to sleep for.

\*/

sleepFor(int milSecs)

{

signal\_code dummySig;

int sleepFor = milSecs;

while(sleepFor)

{

\_os\_sleep(&sleepFor, &dummySig);

}

}

/\*

getCholr():

A method that will get the chlorinated value from the

stable structure using events.

\*/

char getCholr()

{

signal\_code signal;

char ans;

if((errno = \_os\_ev\_wait(stable\_ev\_id,

&stableValue,

&signal,

1,

1)) != 0)

{

printf("P6: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

ans = stable->chlorinated;

if((errno = \_os\_ev\_signal(stable\_ev\_id,

&stableValue,

0)) != 0)

{

printf("P6: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

return ans;

}

/\*

getFlor():

A method that will get the florinated value from the

stable structure using events.

\*/

char getFlor()

{

signal\_code signal;

char ans;

if((errno = \_os\_ev\_wait(stable\_ev\_id,

&stableValue,

&signal,

1,

1)) != 0)

{

printf("P6: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

ans = stable->florinated;

if((errno = \_os\_ev\_signal(stable\_ev\_id,

&stableValue,

0)) != 0)

{

printf("P6: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

return ans;

}

/\*

ready():

A method that will get both values from the stable

structure and see if they

are set to true. Return a boolean based on that result.

\*/

int ready()

{

char C = getCholr();

char F = getFlor();

if((C == 'Y') && (F == 'Y'))

{

return 1;

}

return 0;

}

/\*

empty():

A method that will reset the stable structure using

events, change the water

structure using events and send a message to P7.

\*/

empty()

{

error\_code err;

signal\_code signal;

if((errno = \_os\_ev\_wait(stable\_ev\_id,

&stableValue,

&signal,

1,

1)) != 0)

{

printf("P6: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

stable->chlorinated = 'N';

stable->florinated = 'N';

if((errno = \_os\_ev\_signal(stable\_ev\_id,

&stableValue,

0)) != 0)

{

printf("P6: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

if((errno = \_os\_ev\_wait(water\_ev\_id,

&waterValue,

&signal,

1,

1)) != 0)

{

printf("P5: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterStruct->filtered = 'Y';

waterStruct->sedimented = 'Y';

waterStruct->flocculated = 'Y';

if((errno = \_os\_ev\_signal(water\_ev\_id,

&waterValue,

0)) != 0)

{

printf("P5: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

if(err = (\_os\_send(PID->pidArray[6], 355)) != 0)

{

printf("P6: Failed to send to P7\n");

}

}

main()

{

error\_code err;

u\_int32 milSecs;

signal\_code ReceivedSignal;

alarm\_id MyAlarm;

signal\_code WakeupSignal;

u\_int32 TimeToDelay;

if((err = \_os\_intercept(sig\_handler,

\_glob\_data)) != 0)

{

printf("P6: Failed to attach signal handler\n");

exit(err);

}

WakeupSignal = 356;

TimeToDelay = SENSOR\_FREQUENCY;

if((errno = \_os\_alarm\_cycle(&MyAlarm,

WakeupSignal,

TimeToDelay)) !=0)

{

printf("P6: error creating alarm\n");

exit(errno);

}

milSecs = 0;

while(1)

{

\_os\_sleep(&milSecs, &ReceivedSignal);

if(!linked)

link();

if(ReceivedSignal == 356)

{

if(ready())

{

printf("P6: Detected that system has

treated water.\n");

sleepFor(SLEEP\_FOR);

printf("P6: Sending finish to P7.\n");

empty();

}

}

}

}

**Explanation:**

* P6 is similar to a sensor as in, it will simulate a process that is always checking the stable structure for changes.
* P6 has a cyclic alarm that will wake it up every 3 seconds to check both values in the stable structure using events.
* If both values, ‘chlorinated’ and ‘florinated’ have a value of ‘Y’, then that’s means that the sensors and actuators (P2,P2,P3 and P4) are finished with their work and the water has been stabilized.
* P6 will then reset these values back to ‘N’ and send a message to P7 to continue the execution of the system.

**P7.c**

**Status:** Finished. Working 100%.

**Source Code:**

#include <stdio.h>

#include <signal.h>

#include <types.h>

#include <modes.h>

#include <errno.h>

#include <cglob.h>

#include <stdlib.h>

#include "PIDStruct.h"

#include "WaterStruct.h"

#define PID\_ARRAY\_NAME "PID"

#define WATER\_STRUCT\_NAME "WaterStruct"

/\*

SLEEP\_FOR: amount of milliseconds to sleep for before empyting the water.

\*/

#define SLEEP\_FOR 5000

u\_int32 SleepValue;

mh\_com mod\_head;

char \*ptrPIDName;

char \*ptrWaterName;

struct PIDStruct \*PID;

struct WaterStruct \*waterStruct;

signal\_code DummySignal;

signal\_code dummy\_sig;

event\_id ev\_id;

u\_int32 value;

int linked = 0;

u\_int32 milSecs = 3000;

u\_int16 attr\_rev = (MA\_REENT << 8);

u\_int16 type\_lang = (MT\_DATA << 8);

u\_int16 mem\_size\_pid = STRUCT\_SIZE\_PID;

u\_int16 mem\_size\_water = STRUCT\_SIZE\_WATER;

sig\_handler(signal\_code sig)

{

switch(sig)

{

case 3:

printf("\nP7: Signal to shut down\n");

\_os\_exit(0);

break;

}

\_os\_rte();

}

/\*

link():

A method to link to all the recources needed by the

process.

Once linked, it will set 'linked' to 1 to indicate that

everything is already linked so it wont try to link again.

\*/

link()

{

ptrPIDName = PID\_ARRAY\_NAME;

ptrWaterName = WATER\_STRUCT\_NAME;

errno = \_os\_link(&ptrPIDName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&PID,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P7: %d: Couldnt Link to data pid

module\n", \_procid);

\_os\_exit(errno);

}

errno = \_os\_link(&ptrWaterName,

(mh\_com\*\*)&mod\_head,

(void\*\*)&waterStruct,

&type\_lang,

&attr\_rev);

if(errno != 0)

{

fprintf(stderr, "P7: %d: Couldnt Link to water data

module\n", \_procid);

\_os\_exit(errno);

}

if((errno = \_os\_ev\_link(WATER\_STRUCT\_NAME,

&ev\_id)) != 0)

{

printf("P7: failed to link to water semaphore,

shutting down\n");

\_os\_exit(errno);

}

linked = 1;

}

/\*

sleepFor(int milSecs):

A method that will take in an int value to sleep for. No

signal will interupt its sleep.

milSecs: amount of milliseconds to sleep for.

\*/

sleepFor(int milSecs)

{

signal\_code dummySig;

int sleepFor = milSecs;

while(sleepFor)

{

\_os\_sleep(&sleepFor, &dummySig);

}

}

/\*

emptyWater():

A method that will reset the values in the water

structure by using events.

\*/

emptyWater()

{

signal\_code signal;

printf("P7: Empting.....\n");

sleepFor(SLEEP\_FOR);

if((errno = \_os\_ev\_wait(ev\_id,

&value,

&signal,

1,

1)) != 0)

{

printf("P7: Error waiting for semaphore, shutting

down\n");

\_os\_exit(errno);

}

waterStruct->filtered = 'N';

waterStruct->sedimented = 'N';

waterStruct->flocculated = 'N';

waterStruct->pHLevel = 0;

waterStruct->chlorinationLevel = 0;

waterStruct->fluoridationLevel = 0;

waterStruct->hardWater = 0;

if((errno = \_os\_ev\_signal(ev\_id,

&value,

0)) != 0)

{

printf("P7: Signaling event error, shutting

down\n");

\_os\_exit(errno);

}

}

main()

{

error\_code err;

u\_int32 milSecs;

signal\_code ReceivedSignal;

if((err = \_os\_intercept(sig\_handler,

\_glob\_data)) != 0)

{

printf("P7: Failed to attach signal handler\n");

exit(err);

}

milSecs = 0;

while(1)

{

\_os\_sleep(&milSecs, &ReceivedSignal);

if(!linked)

link();

if(ReceivedSignal == 355)

{

printf("P7: Received signal to empty the water

tank. \n");

emptyWater();

printf("P7: Water has been emptied. System

finished.... \n");

sleepFor(4000);

printf("P7: Sending message to P1 to restart

system. \n");

sleepFor(4000);

printf("\n\n");

if(err = (\_os\_send(PID->pidArray[0], 356)) != 0)

{

printf("P7: Failed to send to P1\n");

}

}

}

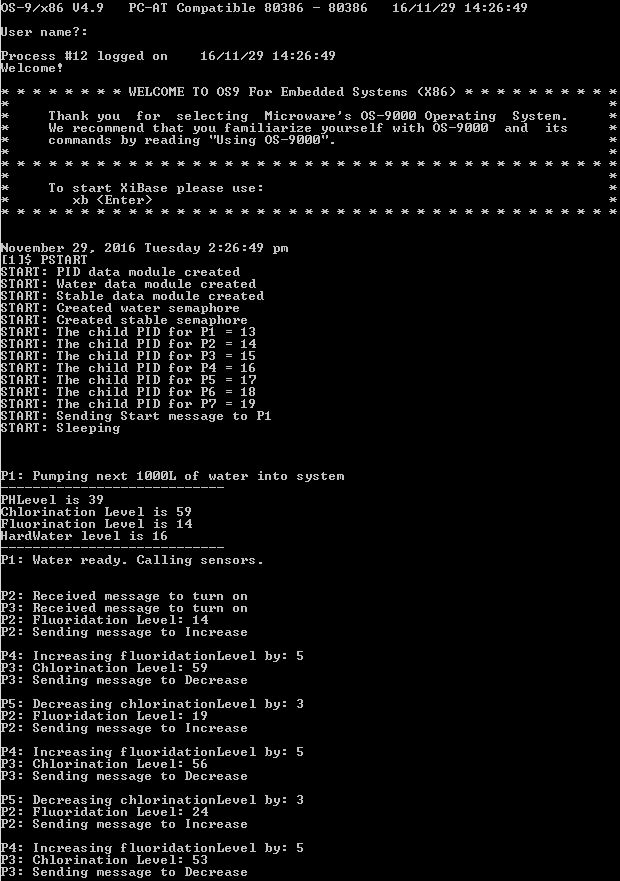
}

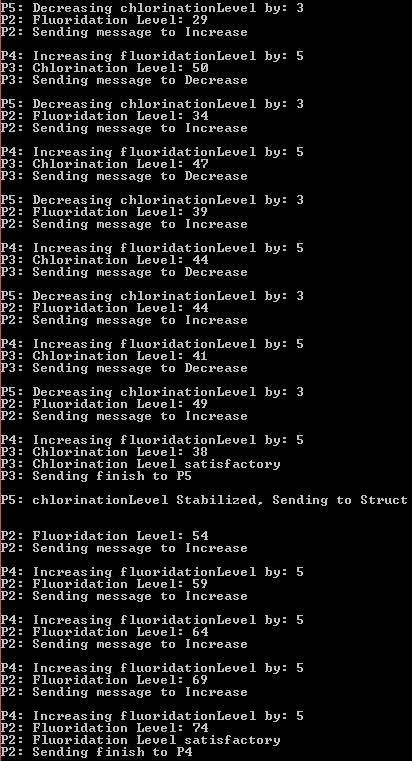
**Explanation:**

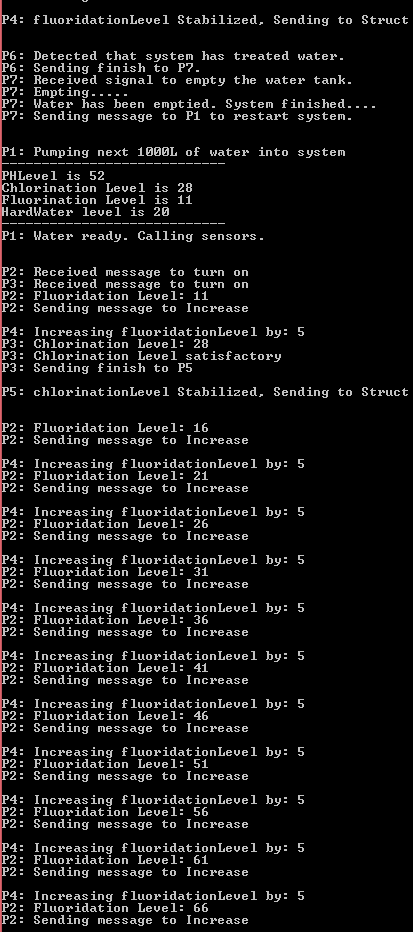
* P7 is to simulate a process that will empty the system of water and any other procedures to finish the systems execution and send a message to P1 to restart the entire system.
* P7 will sleep until it receives a message from P6. This message will mean that the system is finished and P7 can its execute its final procedure.

It will first simulate the emptying of the water by setting all of the water structures values to 0 by using events. It will then sleep for a few seconds and call P1 to restart the system over again.

**System Testing**







November 29, 2016 Tuesday 2:26:49 pm

[1]$ PSTART

START: PID data module created

START: Water data module created

START: Stable data module created

START: Created water semaphore

START: Created stable semaphore

START: The child PID for P1 = 13

START: The child PID for P2 = 14

START: The child PID for P3 = 15

START: The child PID for P4 = 16

START: The child PID for P5 = 17

START: The child PID for P6 = 18

START: The child PID for P7 = 19

START: Sending Start message to P1

START: Sleeping

P1: Pumping next 1000L of water into system

----------------------------

PHLevel is 39

Chlorination Level is 59

Fluorination Level is 14

HardWater level is 16

----------------------------

P1: Water ready. Calling sensors.

P2: Received message to turn on

P3: Received message to turn on

P2: Fluoridation Level: 14

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 59

P3: Sending message to Decrease

P5: Decreasing chlorinationLevel by: 3

P2: Fluoridation Level: 19

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 56

P3: Sending message to Decrease

P5: Decreasing chlorinationLevel by: 3

P2: Fluoridation Level: 24

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 53

P3: Sending message to Decrease

P5: Decreasing chlorinationLevel by: 3

P2: Fluoridation Level: 29

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 50

P3: Sending message to Decrease

P5: Decreasing chlorinationLevel by: 3

P2: Fluoridation Level: 34

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 47

P3: Sending message to Decrease

P5: Decreasing chlorinationLevel by: 3

P2: Fluoridation Level: 39

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 44

P3: Sending message to Decrease

P5: Decreasing chlorinationLevel by: 3

P2: Fluoridation Level: 44

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 41

P3: Sending message to Decrease

P5: Decreasing chlorinationLevel by: 3

P2: Fluoridation Level: 49

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 38

P3: Chlorination Level satisfactory

P3: Sending finish to P5

P5: chlorinationLevel Stabilized, Sending to Struct

P2: Fluoridation Level: 54

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 59

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 64

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 69

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 74

P2: Fluoridation Level satisfactory

P2: Sending finish to P4

P4: fluoridationLevel Stabilized, Sending to Struct

P6: Detected that system has treated water.

P6: Sending finish to P7.

P7: Received signal to empty the water tank.

P7: Empting.....

P7: Water has been emptied. System finished....

P7: Sending message to P1 to restart system.

P1: Pumping next 1000L of water into system

----------------------------

PHLevel is 52

Chlorination Level is 28

Fluorination Level is 11

HardWater level is 20

----------------------------

P1: Water ready. Calling sensors.

P2: Received message to turn on

P3: Received message to turn on

P2: Fluoridation Level: 11

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P3: Chlorination Level: 28

P3: Chlorination Level satisfactory

P3: Sending finish to P5

P5: chlorinationLevel Stabilized, Sending to Struct

P2: Fluoridation Level: 16

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 21

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 26

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 31

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 36

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 41

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 46

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 51

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 56

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

P2: Fluoridation Level: 61

P2: Sending message to Increase

P4: Increasing fluoridationLevel by: 5

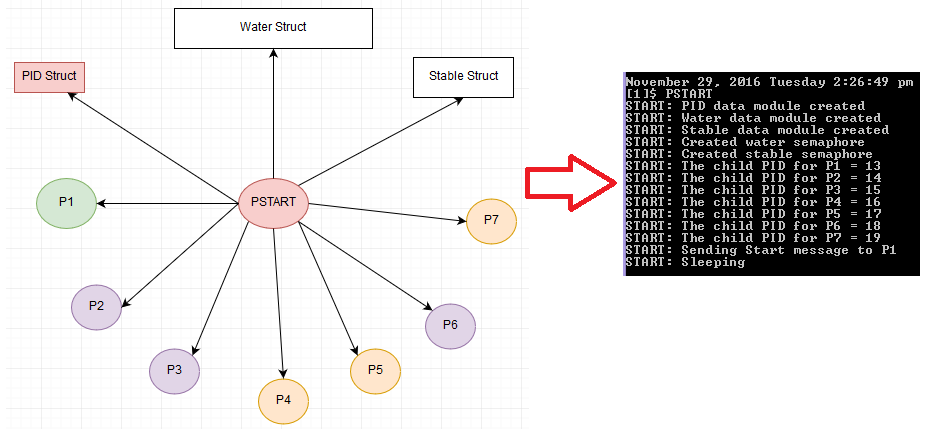
P2: Fluoridation Level: 66

P2: Sending message to Increase

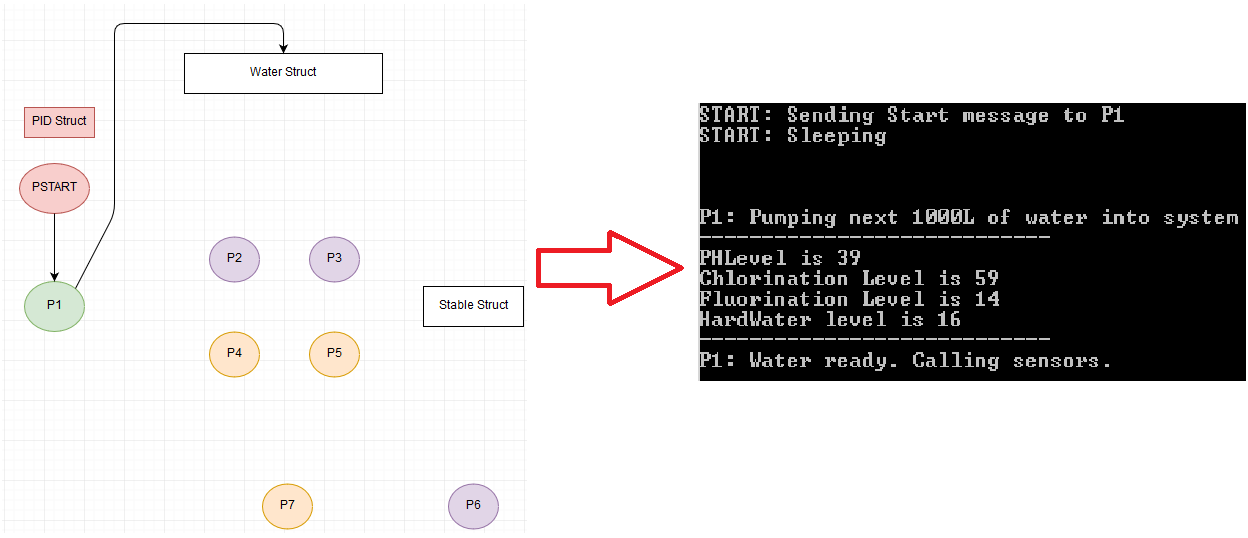
* Please note to hit enter on the console as it stops printing after a few lines.

**Explanation:**

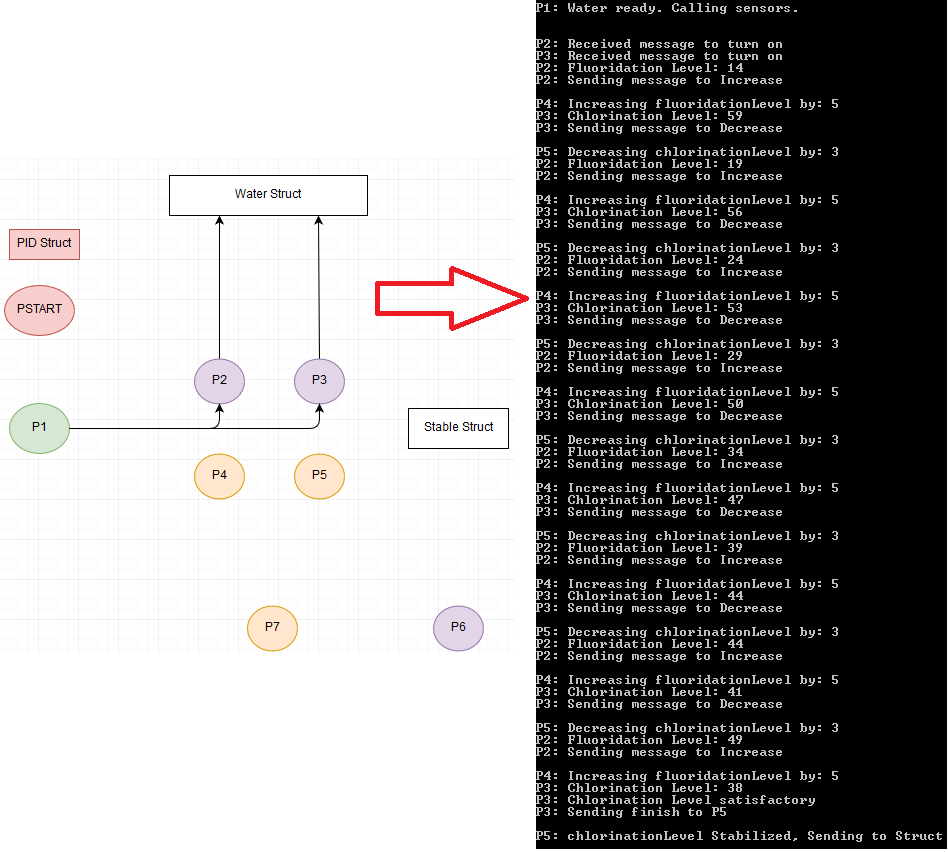
* PSTART is called and it will create the data modules needed, the semaphores/events needed and fork each of the 7 processes. It will then fill in the process ID array in the PID structure for the rest of the processes to use and send P1 a signal to begin the system.



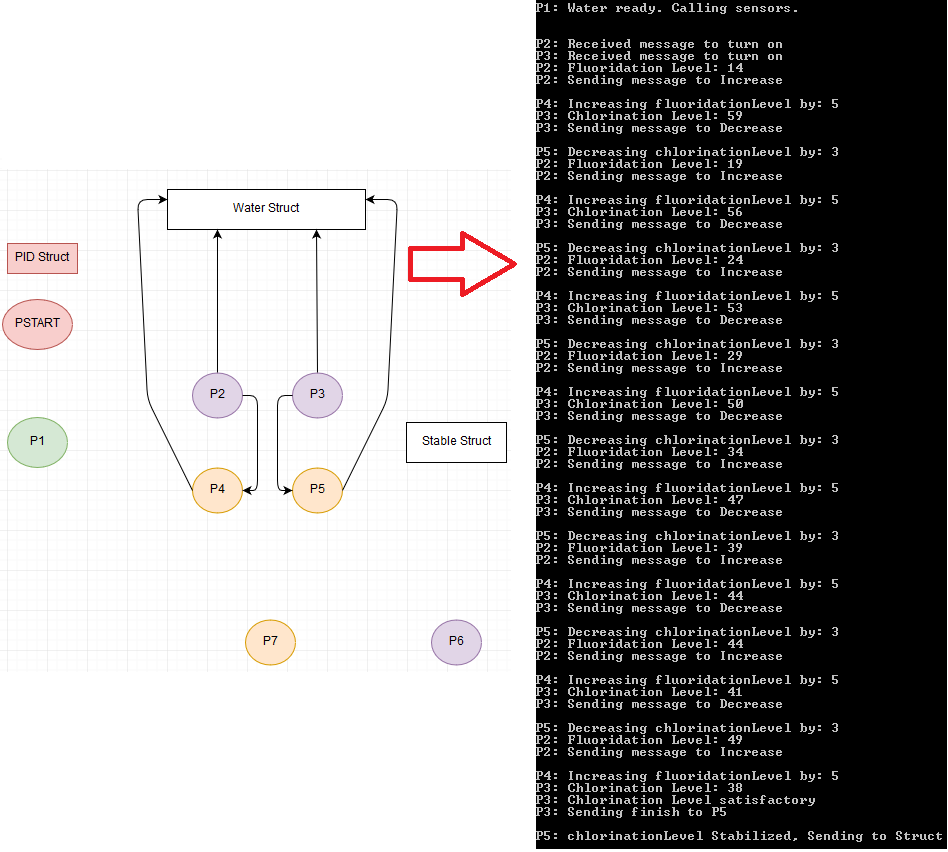
* PSTART sends a message to P1 to begin the system. P1 will assign random values to the water structure using events.



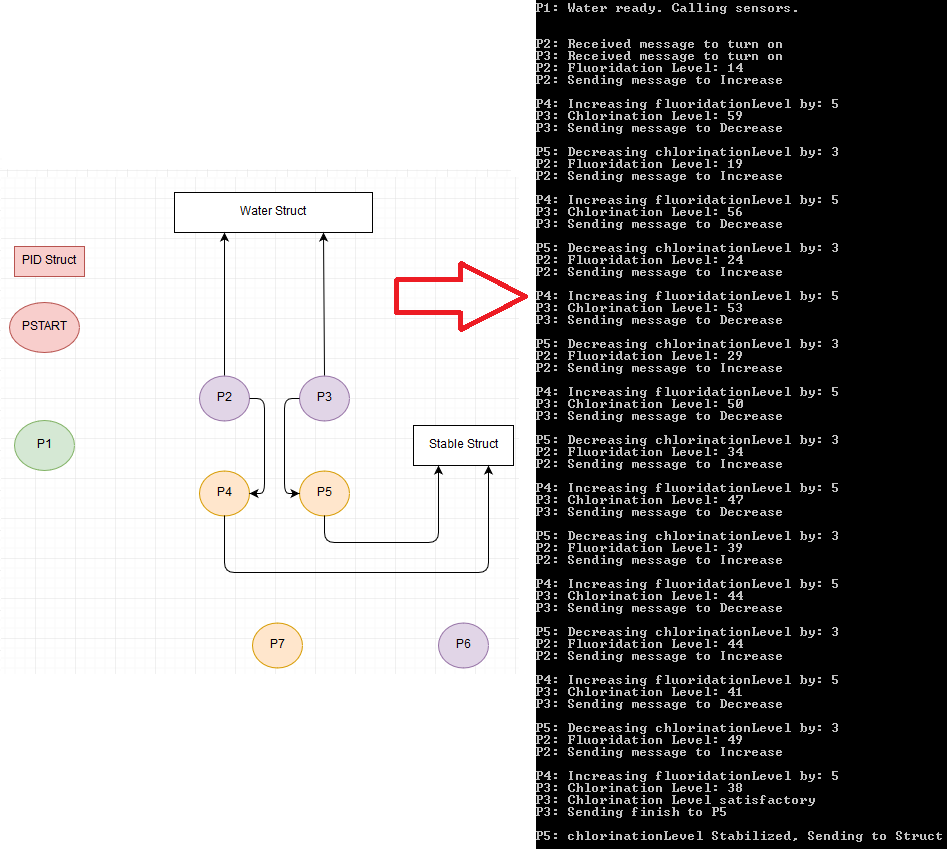
* Once this is done P1 will send a signal to P2 and P3 to turn on and begin sensing the water structure by using events.



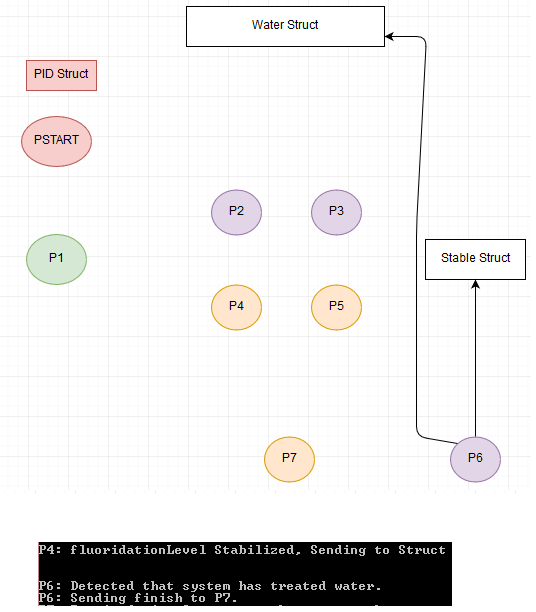
* P2 and P3 will start reading the water structure using events and if the values are unsatisfactory they will send messages to P4 and P5 respectfully to change the values.



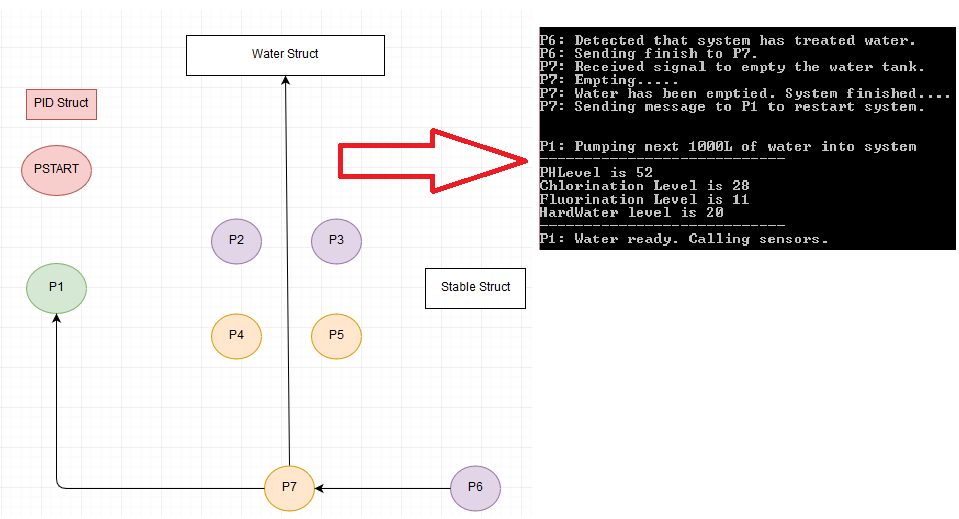
* P4 and P5 will then wake up and change the values on the water structure using events based on the message received from P2 and P3.
* Once the water structures values are satisfactory, P2 and P3 will send a finish message to P4 and P5 respectively, and turn themselves off. P4 and P5 will in turn change the Booleans on the stable structure to positive by using events.



* P6 is always checking the stable structure to see if both Booleans are positive. If they are then it will reset them back to negative and send a signal to P7.



* P6 will set the water structures Booleans to true to indicate that it if finished by using events and send a signal to P7.
* P7, once received the signal, will set the water structures values to 0 and Booleans to negative and call P1 to restart the system again.



* The system will then restart and P1 will enter a new batch of water into the system and call the sensors to turn on.

