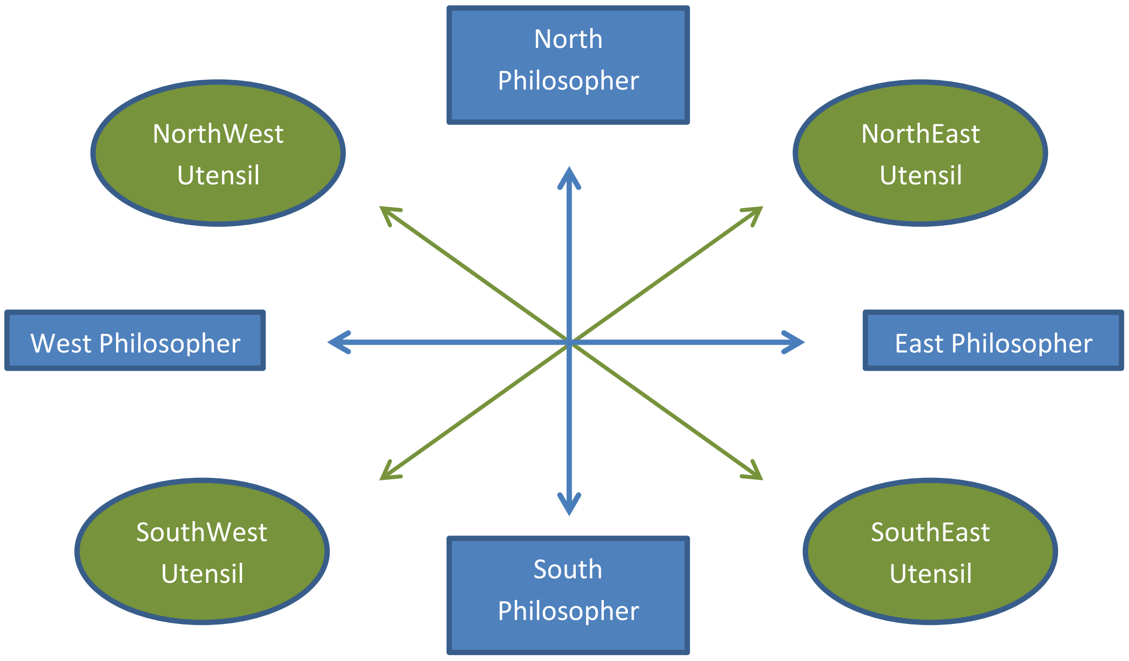
Lovelace, Joe
Dickman, Tom
Zerhusen, Ben
Steller, Andrew

Embedded Systems Project 3: Dining Philosophers

Problem

The dining philosophers' problem encompasses the issues of resource sharing in operating systems. This problem takes four philosophers, north, south, east, and west, and requires them to switch between thinking and eating respectively. When eating the philosopher must acquire two forks, on both his left and right hands respectively. There however are only four forks on the table, one between each philosopher, and therefore they must be shared in a manner that none of the philosophers are starved because of waiting to eat for too long.



Implementation

Our solution to this problem is to use a switching algorithm to change the order in which the philosophers acquire their utensils (resources). The basics of the algorithm are that when a philosopher grabs for the left fork first he will then look for the right fork. If the right fork is locked by another philosopher then the philosopher will release the left fork and change to look for the right fork first. This allows the philosopher to still maintain a place in the queue but not lock up resources unnecessarily and starve other philosophers.

In our implementation the utensils are represented by semaphores that are managed by the µC/OSII kernel. When each philosopher is created in the initiPhilosophers() function there "left" and "right" members are assigned to one of the four semaphores (utensils) based on their position at the table. Each philosopher also has members corresponding to their state () and their count (number of times they have eaten). Using carefully controlled printf() statements we are able to print the current status of the philosophers as they change state into the console window for the NIOSII. Carefully examining the output in the console window reveals that none of the philosophers are being starved and the resources are being well shared between each of them.

Testing Strategy

The testing strategy was largely driven by the need for independent testing. Because there is only one main file with a handful of functions that interact very closely, we did not have to worry about testing individual modules and the connections between them. The initPhilosophers() function sets up each of the four philosophers and their utensils, so it was very easy to test. The run() function was simply an infinite loop that runs until the process is stopped. In this function the dine() function is called, which is where the control of the resources occurs. In order to test the functionality of this function we had to use the printf() function to output status messages to the console. Once completed, the top level project was tested by running it on the board. Due to the fact that there is no easy way to simulate an embedded processor application, we did not perform simulation in any program external to the board.

Team Member Contributions

Name	Contributions
Tom Dickman	Code writer and debugger
Joe Lovelace	Code debugger and tester
Andrew Steller	Final report writer
Ben Zerhusen	Code tester, final report editor

Console Output

```
N is hungry
N I'm eating. I've eaten 1 times.
E is hungry
S is hungry
S I'm eating. I've eaten 1 times.
W is hungry
N I'm done eating.
E is hungry
E Couldn't get second fork, swapping forks.
S I'm done eating.
W is hungry
W I'm eating. I've eaten 1 times.
```

Lovelace, Joe
Dickman, Tom
Zerhusen, Ben
Steller, Andrew

```
N is hungry
E is hungry
E I'm eating. I've eaten 1 times.
S is hungry
W I'm done eating.
N is hungry
N Couldn't get second fork, swapping forks.
E I'm done eating.
S is hungry
S I'm eating. I've eaten 2 times.
W is hungry
N is hungry
N I'm eating. I've eaten 2 times.
E is hungry
S I'm done eating.
W is hungry
W Couldn't get second fork, swapping forks.
N I'm done eating.
E is hungry
E I'm eating. I've eaten 2 times.
S is hungry
W is hungry
W I'm eating. I've eaten 2 times.
N is hungry
E I'm done eating.
S is hungry
S Couldn't get second fork, swapping forks.
W I'm done eating.
N is hungry
N I'm eating. I've eaten 3 times.
E is hungry
E Couldn't get second fork, swapping forks.
S is hungry
S I'm eating. I've eaten 3 times.
W is hungry
N I'm done eating.
E is hungry
```

Lovelace, Joe
Dickman, Tom
Zerhusen, Ben
Steller, Andrew

```
E Couldn't get second fork, swapping forks.
S I'm done eating.
W is hungry
W I'm eating. I've eaten 3 times.
N is hungry
N Couldn't get second fork, swapping forks.
E is hungry
E I'm eating. I've eaten 3 times.
S is hungry
W I'm done eating.
N is hungry
N Couldn't get second fork, swapping forks.
E I'm done eating.
S is hungry
S I'm eating. I've eaten 4 times.
W is hungry
W Couldn't get second fork, swapping forks.
N is hungry
N I'm eating. I've eaten 4 times.
E is hungry
S I'm done eating.
W is hungry
W Couldn't get second fork, swapping forks.
N I'm done eating.
E is hungry
E I'm eating. I've eaten 4 times.
S is hungry
S Couldn't get second fork, swapping forks.
W is hungry
W I'm eating. I've eaten 4 times.
N is hungry
E I'm done eating.
S is hungry
S Couldn't get second fork, swapping forks.
W I'm done eating.
N is hungry
N I'm eating. I've eaten 5 times.
```

Lovelace, Joe
Dickman, Tom
Zerhusen, Ben
Steller, Andrew

```
E is hungry
E Couldn't get second fork, swapping forks.
S is hungry
S I'm eating. I've eaten 5 times.
W is hungry
N I'm done eating.
E is hungry
E Couldn't get second fork, swapping forks.
S I'm done eating.
W is hungry
W I'm eating. I've eaten 5 times.
N is hungry
N Couldn't get second fork, swapping forks.
E is hungry
E I'm eating. I've eaten 5 times.
S is hungry
W I'm done eating.
N is hungry
N Couldn't get second fork, swapping forks.
E I'm done eating.
S is hungry
S I'm eating. I've eaten 6 times.
W is hungry
W Couldn't get second fork, swapping forks.
N is hungry
N I'm eating. I've eaten 6 times.
E is hungry
S I'm done eating.
W is hungry
W Couldn't get second fork, swapping forks.
N I'm done eating.
E is hungry
E I'm eating. I've eaten 6 times.
S is hungry
S Couldn't get second fork, swapping forks.
W is hungry
W I'm eating. I've eaten 6 times.
```

Lovelace, Joe
Dickman, Tom
Zerhusen, Ben
Steller, Andrew

```
N is hungry
E I'm done eating.
S is hungry
S Couldn't get second fork, swapping forks.
W I'm done eating.
N is hungry
N I'm eating. I've eaten 7 times.
E is hungry
E Couldn't get second fork, swapping forks.
S is hungry
S I'm eating. I've eaten 7 times.
W is hungry
N I'm done eating.
E is hungry
E Couldn't get second fork, swapping forks.
S I'm done eating.
W is hungry
W I'm eating. I've eaten 7 times.
N is hungry
N Couldn't get second fork, swapping forks.
E is hungry
E I'm eating. I've eaten 7 times.
S is hungry
W I'm done eating.
N is hungry
N Couldn't get second fork, swapping forks.
E I'm done eating.
S is hungry
S I'm eating. I've eaten 8 times.
W is hungry
W Couldn't get second fork, swapping forks.
N is hungry
N I'm eating. I've eaten 8 times.
E is hungry
S I'm done eating.
W is hungry
W Couldn't get second fork, swapping forks.
```

Lovelace, Joe
Dickman, Tom
Zerhusen, Ben
Steller, Andrew

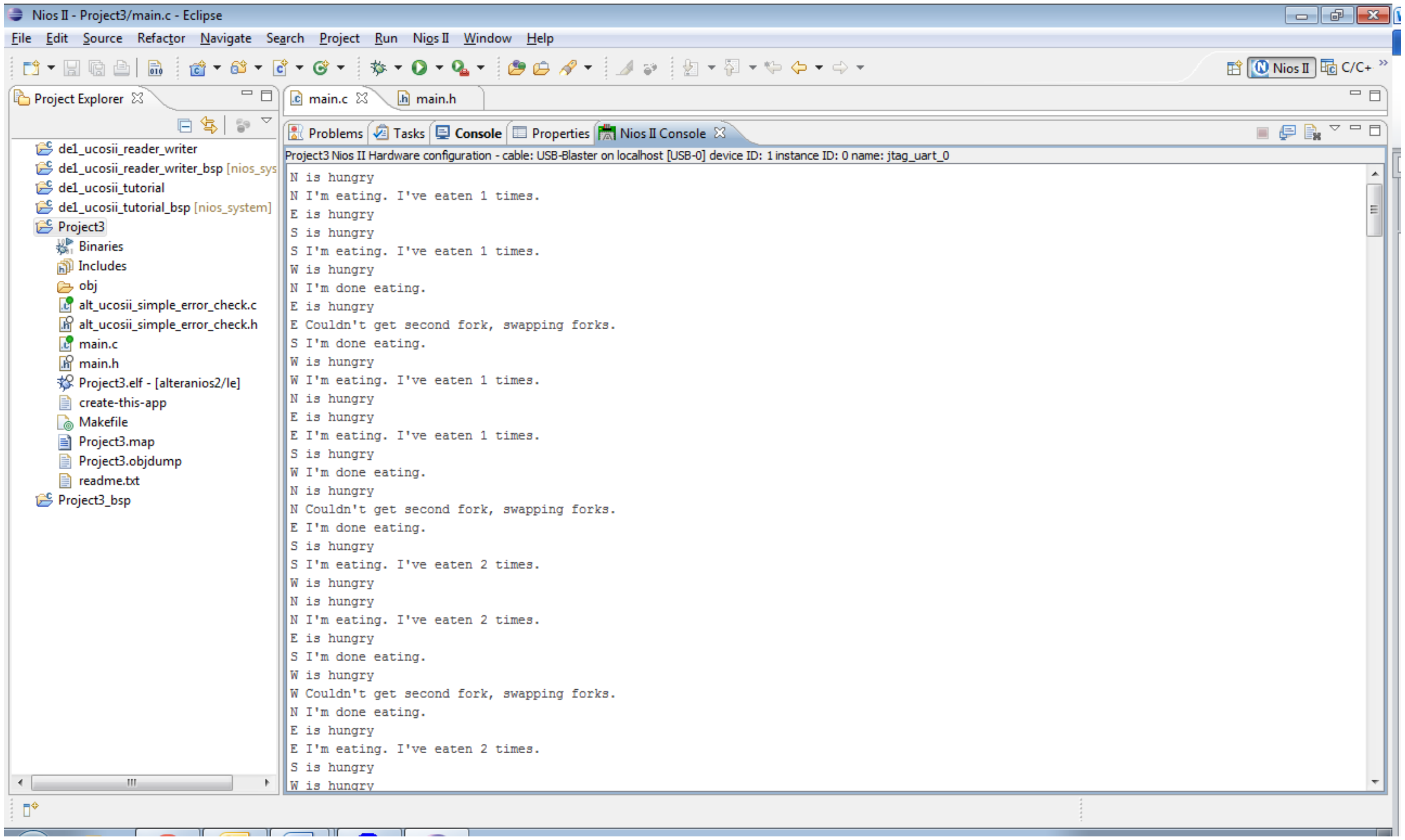
```
N I'm done eating.
E is hungry
E I'm eating. I've eaten 8 times.
S is hungry
S Couldn't get second fork, swapping forks.
W is hungry
W I'm eating. I've eaten 8 times.
N is hungry
E I'm done eating.
S is hungry
S Couldn't get second fork, swapping forks.
W I'm done eating.
N is hungry
N I'm eating. I've eaten 9 times.
E is hungry
E Couldn't get second fork, swapping forks.
S is hungry
S I'm eating. I've eaten 9 times.
W is hungry
N I'm done eating.
E is hungry
E Couldn't get second fork, swapping forks.
S I'm done eating.
W is hungry
W I'm eating. I've eaten 9 times.
N is hungry
N Couldn't get second fork, swapping forks.
E is hungry
E I'm eating. I've eaten 9 times.
S is hungry
W I'm done eating.
N is hungry
N Couldn't get second fork, swapping forks.
E I'm done eating.
S is hungry
S I'm eating. I've eaten 10 times.
W is hungry
```

Lovelace, Joe
Dickman, Tom
Zerhusen, Ben
Steller, Andrew

```
W Couldn't get second fork, swapping forks.
N is hungry
N I'm eating. I've eaten 10 times.
E is hungry
S I'm done eating.
W is hungry
W Couldn't get second fork, swapping forks.
N I'm done eating.
E is hungry
E I'm eating. I've eaten 10 times.
S is hungry
S Couldn't get second fork, swapping forks.
W is hungry
W I'm eating. I've eaten 10 times.
…
```

Lovelace, Joe
Dickman, Tom
Zerhusen, Ben
Steller, Andrew

Output Snapshot


Nios II - Project3/main.c - Eclipse

Project3 Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart_0

```
N is hungry
N I'm eating. I've eaten 1 times.
E is hungry
S is hungry
S I'm eating. I've eaten 1 times.
W is hungry
N I'm done eating.
E is hungry
E Couldn't get second fork, swapping forks.
S I'm done eating.
W is hungry
W I'm eating. I've eaten 1 times.
N is hungry
E is hungry
E I'm eating. I've eaten 1 times.
S is hungry
W I'm done eating.
N is hungry
N Couldn't get second fork, swapping forks.
E I'm done eating.
S is hungry
S I'm eating. I've eaten 2 times.
W is hungry
N is hungry
N I'm eating. I've eaten 2 times.
E is hungry
S I'm done eating.
W is hungry
W Couldn't get second fork, swapping forks.
N I'm done eating.
E is hungry
E I'm eating. I've eaten 2 times.
S is hungry
W is hungry
```

```c
#ifndef ENUMERATIONS_H_
#define ENUMERATIONS_H_

#define TASK_STACKSIZE 2048
#define NUM_PHILOSOPHERS 4

#endif /* ENUMERATIONS_H_ */
```

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "includes.h"
#include "alt_ucosii_simple_error_check.h"
#include "main.h"

OS_STK stack[NUM_PHILOSOPHERS][TASK_STACKSIZE];

// Create our semaphores!
OS_EVENT *utensil[NUM_PHILOSOPHERS];

typedef struct {
    char name;
    OS_EVENT *left;
    OS_EVENT *right;
    int state;
    int count;
} philosopher;
philosopher philosophers[NUM_PHILOSOPHERS];

void dine(philosopher *thePhilosopher)
{
    INT8U return_code;
    // Try to get lock with really short timeout
    OSSemPend(thePhilosopher->left, 1, &return_code);
    if (return_code != OS_TIMEOUT) {
        // We got the fork!!
        OSSemPend(thePhilosopher->right, 1, &return_code);
        if (return_code != OS_TIMEOUT) { // Success
            thePhilosopher->count++;
            printf("%c I'm eating. I've eaten %d times.\n", thePhilosopher->name,
thePhilosopher->count);
            OSTimeDlyHMSM(0,0,5,0); // Sleep 5s
            printf("%c I'm done eating.\n", thePhilosopher->name);
            OSSemPost(thePhilosopher->right);
            OSSemPost(thePhilosopher->left);
        } else { // Failure
            OSSemPost(thePhilosopher->left);
            printf("%c Couldn't get second fork, swapping forks.\n",
thePhilosopher->name);
            OS_EVENT *temp;
            temp = thePhilosopher->left;
            thePhilosopher->left = thePhilosopher->right;
            thePhilosopher->right = temp;
        }
    }
}

void run(philosopher *thePhilosopher)
{
    while(1)
    {
        OSTimeDlyHMSM(0,0,5,0); // Sleep 5s
        printf("%c is hungry\n", thePhilosopher->name);
        dine(thePhilosopher);
    }
}
```

```c
void philosopherLoop(void *pdata)
{
    philosopher *thePhilosopher = (philosopher*)pdata;
    run(thePhilosopher);
}

// Initializes the needed semaphores, and sets the left and right pointers
void initPhilosophers()
{
    INT8U return_code = OS_NO_ERR;

    char names[4] = {'N', 'E', 'S', 'W'};

    // Initialize semaphores
    int i;
    for (i=0; i<NUM_PHILOSOPHERS; i++) {
        utensil[i] = OSSemCreate(1);
    }

    // Initialize philosophers
    for (i=0; i<NUM_PHILOSOPHERS; i++) {
        philosophers[i].left = utensil[i];
        philosophers[i].right = utensil[(i+1)%NUM_PHILOSOPHERS];
        philosophers[i].state = 0;
        philosophers[i].count = 0;
        philosophers[i].name = names[i];
        return_code = OSTaskCreate(philosopherLoop, &philosophers[i], (void*)&stack[i]
[TASK_STACKSIZE-1], i);
        alt_ucosii_check_return_code(return_code);
    }
}

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

    OSStart();

    return 0;
}
```