CS559: Computer Systems Lab - 1

Assignment - 2
Date - 17.08.2018

In this assignment you need to develop a compiler for the following language using lex/yacc or any other equivalent.

The following is the description of the constructs supported in the multi-threaded language. At the top of the hierarchy is a *system*. A system is defined as a set of communicating processes. These processes communicate through messages which are declared globally across the system. We now define what a process is. A process has a set of local declaration in a code block and has a start node and an end node which are markers of the process. It supports the following constructs:

- BLOCK – A block of code
- DECISION Construct
- Message Send Construct
- Message Receive Construct
- GOTO construct

The block of code is a piece of C-code which may or may not use the variables declared locally in the process. Every block is marked by an identifier which uniquely identifies the code block in the process space. A decision construct decides on the control flow of the language. The GOTO construct is an unconditional jump construct. All jumps are within the process space. The messages send/receive constructs are used to exchange messages globally across the various processes.

We now define the syntax of the specification: (All keywords are in CAPS)

```
SYSTEM sysid;
//sysid – System Identifier

SIGNAL   sigid1, sigid2, sigid3;
//sigid – Signal Identifier

PROCESS pid1, pid2, pid3;
//pid – Process Identifier

PROCESSDEF pid1
//Process Definition
START
PROCESSBODY
```

END

PROCESSBODY as defined earlier is a collection of the four kinds of constructs. We define these constructs in more detail now:

<u>BLOCK construct:</u>

```
BLOCK blockid          //Block identifier
START
Standard C code - Anything
END
```

If the blockidentifier blockid is not defined in the process scope, it is an error condition. C code is standard C-code.

<u>DECISION construct:</u>

```
DECISION decisionid   //Decision identifier
IF condition
THEN
GOTO blockid_true
```
Optional - ELSE
Optional - GOTO blockid_false

If the blockidentifiers blockid_false and blockid_true are not defined in the process scope, it is an error condition.

<u>UNCONDITIONAL JUMP:</u>

This is a jump with just one GOTO statement. The syntax is:

```
UNCONDDEC undecid //Identifier to this block
GOTO blockidentifier
```

If the blockidentifier is not defined in the process scope, it is an error condition.

<u>MESSAGE Send and Receive</u>:

```
SEND (messageid)
RECEIVE (messageid)
```

Messageid – should be in the list of signals defined in the global scope otherwise it is an error.

All the identifiers follow standard C identifier based rules.

Translation into Threaded C-code

- Each of the process is translated to threads in Unix/Linux.
- Each send/receive message is to be done through named pipes which are blocking.

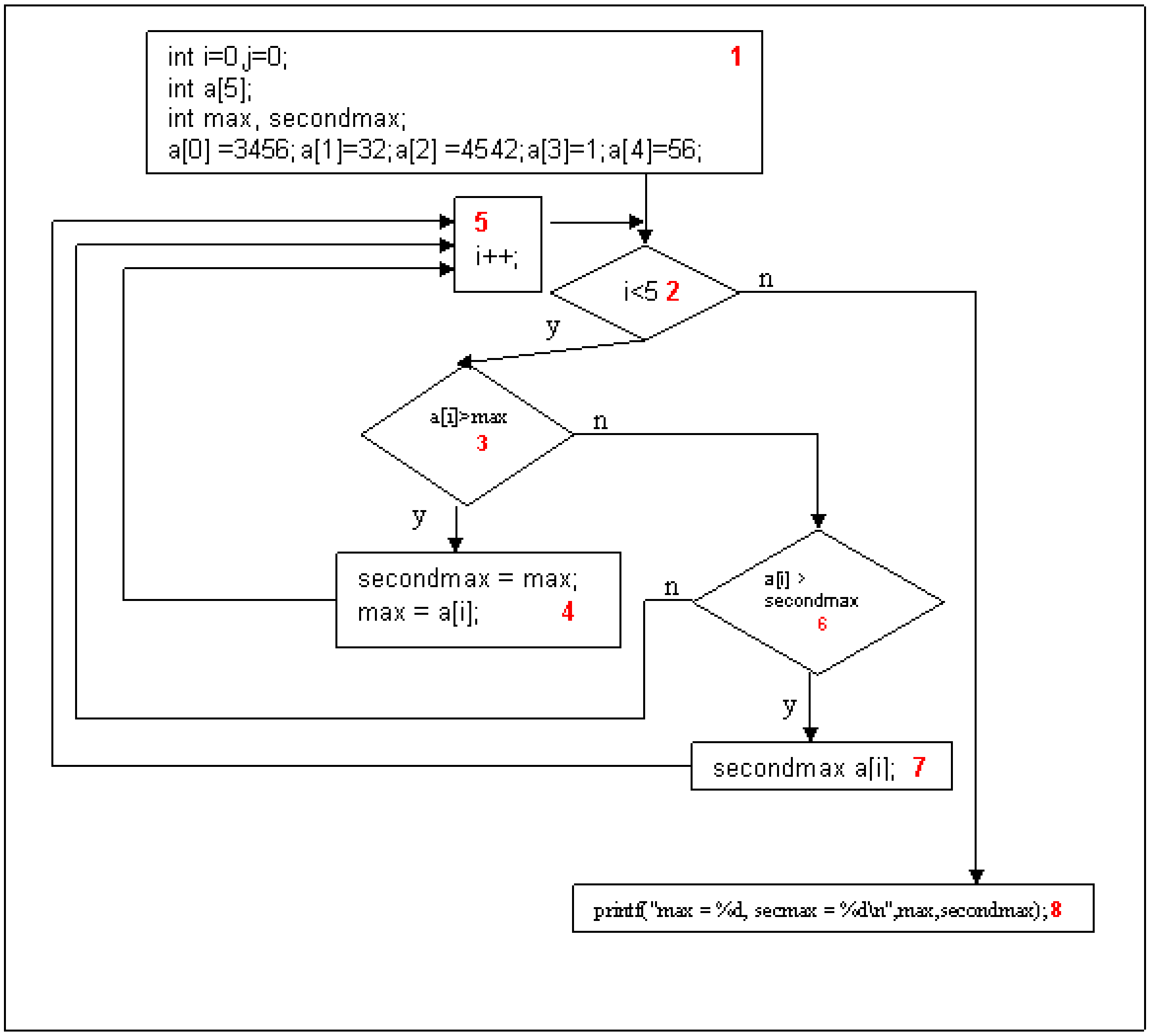
There are two stages in the assignment:
Stage – I: There is just one process and you have to generate sequential C-code from the specification.

Stage – II: There are multiple processes and there are message send/receives.

The SYSTEM/PROCESS/SIGNAL lines are redundant in this sequential code generation process. They will be used in the later stage of the assignment.

Message Passing will be described in more detail in the second part of the assignment.
A more detailed figurative example:

//Finding the largest and the second largest in a set of numbers



Actual C-Code which depicts the second maximum and maximum finding algorithm.

```
#include<stdio.h>
main(){

int i=0,j=0;
int a[5];
int max, secondmax;

a[0] =3456;
a[1]=32;
a[2] =4542;
a[3]=1;
a[4]=56;

for(i=0;i<5;i++){

        if(a[i]>max) {secondmax = max; max = a[i];}
        else {
        if(a[i] > secondmax) secondmax = max;
        }
}
}
```

The Actual C- code.

The C-code with unstructured GOTO statements is *acceptable* but *NOT desirable*. There will be some penalty.

Input for Stage - I: SecondMax and Max

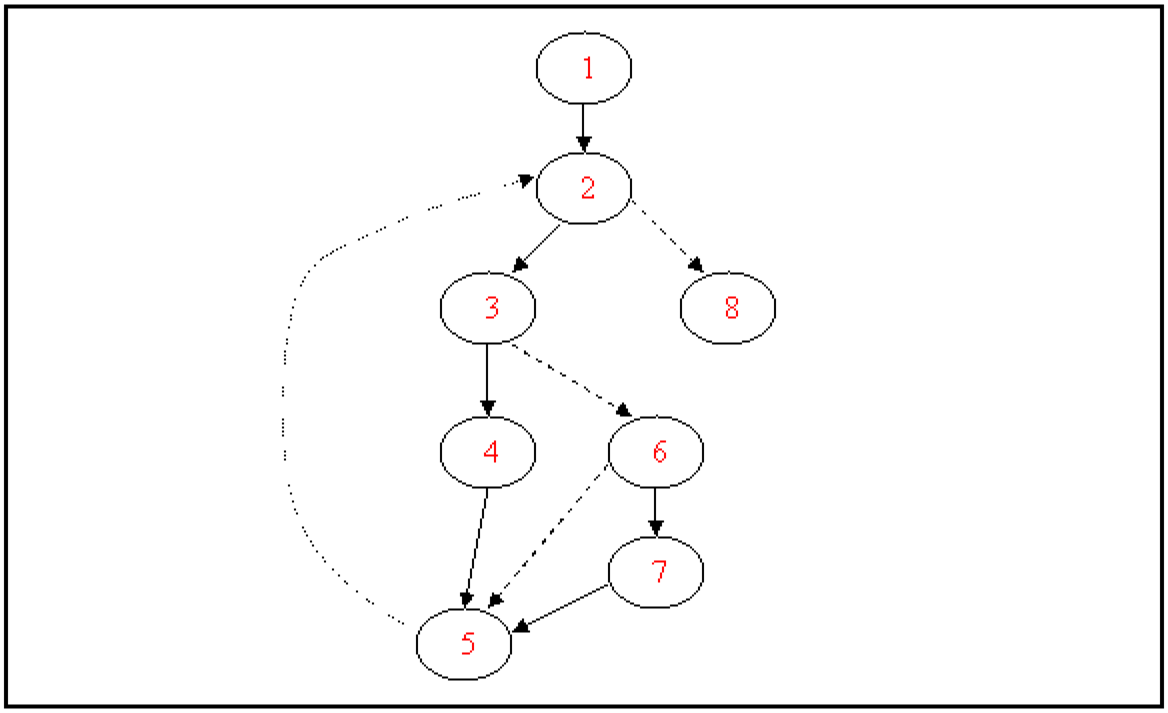
```
SYSTEM mysystem;
SIGNAL s1;
PROCESS pid1;

PROCESSDEF pid1
START

BLOCK myblock1
START
int i=0,j=0;
int a[5];
int max, secondmax;
a[0] =3456;a[1]=32;a[2] =4542;a[3]=1;a[4]=56;
END

DECISION mydec1
IF i < 0
THEN
GOTO mydec2
ELSE
GOTO myblock8


DECISION mydec2
IF a[i] > max
```

```
THEN
GOTO myblock4
ELSE
GOTO mydec3



BLOCK myblock4
START
secondmax = max; max = a[i];
END

DECISION mydec3
IF a[i] > secondmax
THEN
GOTO myblock7
ELSE
GOTO myblock5

BLOCK myblock5
START
i=i++;
GOTO mydec1;
END

BLOCK myblock7
START
secondmax a[i];
GOTO myblock5;
END

BLOCK myblock8
START
printf("max = %d, secmax = %d\n",max,secondmax);
END


END
```

Hint: Solution Methodology

We first construct a graph out of it using a DFS of the parse tree. The IDs marked in RED are assigned to the various blocks in the specification according to topological sorted order.

The tree constructed looks like this:



<u>The code generated for the above example will be as follows:</u>

```
C-Code for node 1;
If(condition of Node 2){
  If(condition of Node 3){
            C- Code for Node 4;
  } else if(condition for Node 6){
            C- Code for Node 7;
  }
C-code for node 5;
//Generate a goto here
goto node2;
} else {

C- code for Node 8.

}
```