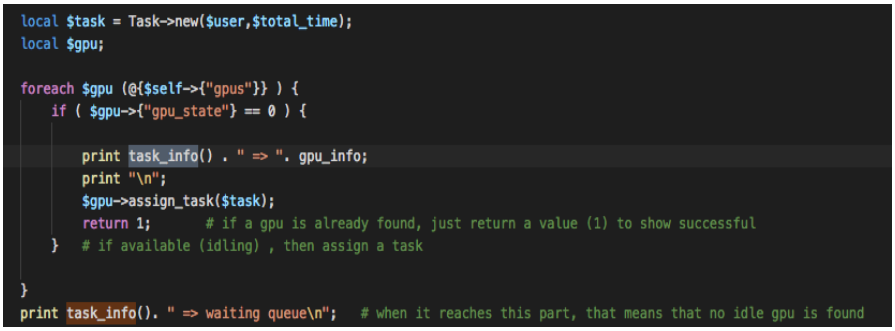
Nigel Nicholas
1155088791

Assignment 3 Report

1. Dynamic Scoping in Task2

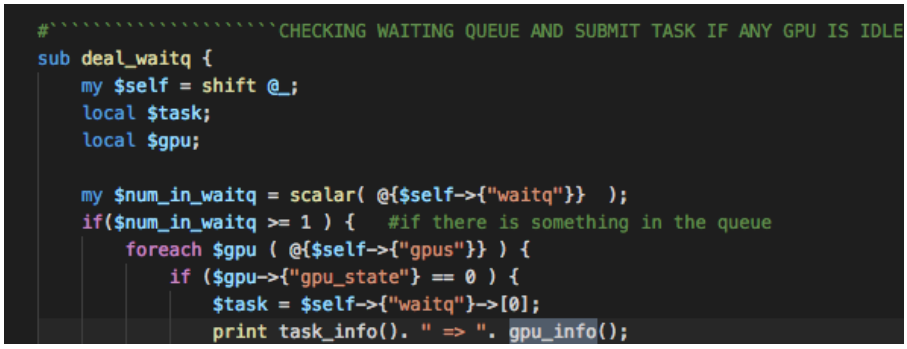
In Task 2, there are 3 functions provided in the assignment package that requires us to use it. These subroutines are named: ***task_info ,task_attr*** and ***gpu_info.*** The following snapshots will show how I use dynamic scoping in relation to these 3 functions.



```
local $task = Task->new($user,$total_time);
local $gpu;

foreach $gpu (@{$self->{"gpus"}} ) {
    if ( $gpu->{"gpu_state"} == 0 ) {

        print task_info() . " => ". gpu_info;
        print "\n";
        $gpu->assign_task($task);
        return 1;       # if a gpu is already found, just return a value (1) to show successful
    }   # if available (idling) , then assign a task

}
print task_info(). " => waiting queue\n";   # when it reaches this part, that means that no idle gpu is found
```

As you see here, unlike other packages in task2 or the entire task1, I use local to represent the task and gpu. Using local allows us to utilize perl's dynamic scoping. Because if we take the function task_info, we see that the returned function doesn't have any parameter passed, and immediately use the variable name, in this case ***$task****.*

The few following pictures is how I use the 2 other subroutines as mentioned above:



```
#`````````````````````CHECKING WAITING QUEUE AND SUBMIT TASK IF ANY GPU IS IDLE
sub deal_waitq {
    my $self = shift @_;
    local $task;
    local $gpu;

    my $num_in_waitq = scalar( @{$self->{"waitq"}}  );
    if($num_in_waitq >= 1 ) {   #if there is something in the queue
        foreach $gpu ( @{$self->{"gpus"}} ) {
            if ($gpu->{"gpu_state"} == 0 ) {
                $task = $self->{"waitq"}->[0];
                print task_info(). " => ". gpu_info();
```

Using gpu_info as highlighted. This time, we use the variable $gpu whose earlier we already declare as local variable.

```
foreach $task( @{$self->{"waitq"}} ) {  # check also for the list in waitq
    my ($name, $pid, $total_time) = task_attr();
    print "        ";   # the spacing first
    print "wait";
    print "   ";
    print $name;
    print "    ";
    print $pid;
    print "      ";
    print $total_time;
```

*This time, we use task_attr as highlighted. The subroutine(function) returns a list of attributes such as , name, id and total time. *note: the $task in the picture was declared local. Refer to the code in Server.pm.*

2. Advantage and Disadvantages of Dynamic Scoping Relative to Lexical Scoping

2.1 Advantage:

The advantage of using dynamic scoping is that, just like the function provided and the usage, we don't need to declare which object's attribute is it that we return. We immediately access the attribute through the use of reserved name ***local***. This makes it easier for us if we already declare a variable local and in the function we call, we just refer to this variable. In the other function, we don't need to use " ***shift @_"*** or parameters because the other function already know what we're referring to. Thus making It easier for us if we are using for loop and inside the for loop we call the function that is dynamically scoped.

2.2 Disadvantage:

The apparent disadvantage of using dynamic scoping is mostly when doing the coding. Because the variable name in dynamic scoping is fixed, we have to always have the same name of variable in the 2 functions. For example, because we have ***$task*** in the function ***task_info,*** then we also need to have the name ***$task*** in our calling function. This causes inflexibility in naming convention unlike using parameter passing.

Another disadvantage is that it is difficult to read our code again because we have to trace which part of the code calls the function through dynamic scoping because 2 different part of function call produces different result. Unlike lexical scoping, we can just see immediately the variable if it is in the same code block declared or not, if not, we can check the outer block that encapsulates this part, and so on until you can see whether it's a global variable or not.