Runtime Environments

Sudakshina Dutta

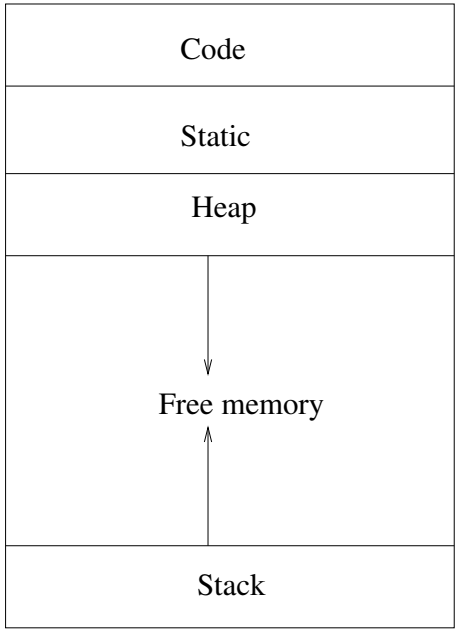
IIT Goa

26^{th} April, 2022

Run-Time Environments

- A compiler must accurately implement the abstraction of the programming language
  1. names
  2. scopes
  3. bindings
  4. data types
  5. operators
  6. parameters
  7. flow-of-control
- Compiler creates and manages run-time environment in which target program gets executed
- Deals with
  1. storage allocation
  2. accessing variables
  3. passing parameters
  4. interface to operating system

Subdivision of run-time memory of object program



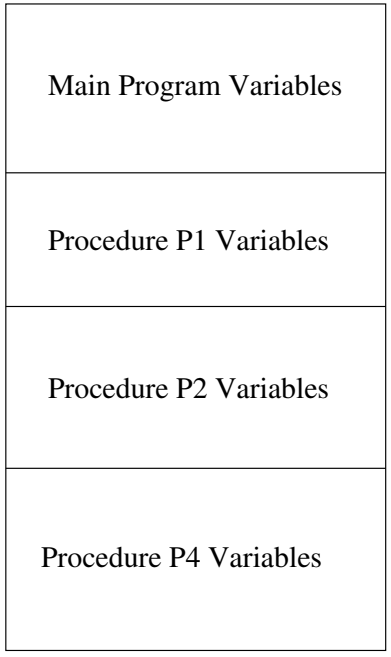
As taught in OS:

Stack
Heap
Static/Global
Code

- Logical address space divided by a compiler
- Size of the target code is static and it is kept in the "Code" section.
- Size of program objects, such as global constants, information generated by compiler is kept in "Static"
- Stack and heap both are dynamic. Their size can change. Former is used for keeping activation records.

Static storage allocation

- Compiler makes the decision regarding storage allocation by looking only at the program text
- No stack/heap allocation; no overheads
- Variable access is fast since addresses are known at compile time
- Recursion is not allowed
- Example - Fortran



Dynamic storage allocation

- Storage allocated at run-time
  - Stack storage : names local to a procedure are kept. Also called "run-time stack"
  - Heap storage : Data that stays when the procedure returns
- Variable access is slow (compared to static allocation) since addresses are accessed through the stack/heap pointer
- Recursion
- Example - C

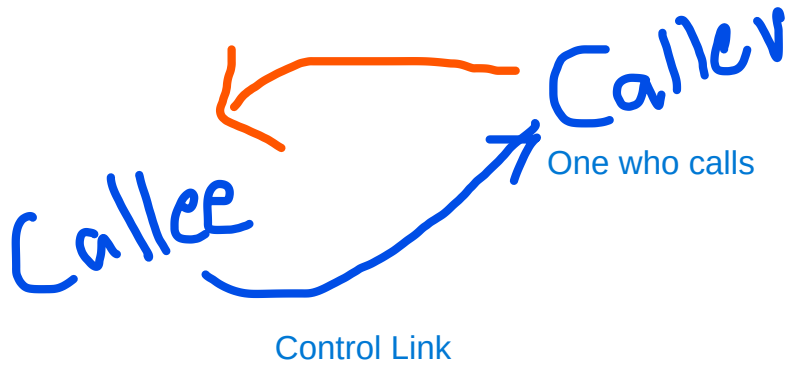
Activation record

- An activation record is used to store information about the status of the machine, such as the value of program counters, registers at time of procedure calls
- When control returns from the call, activation of the calling procedure is restarted to the point immediately after the call
- The data which belongs to the activation is allocated in the activation along with other information

Activation record

Actual parameters
Returned values
Control link
Access link
Saved machine status
Local data
Temporaries

PCB in OS course?



- **Temporaries** for storing the result of evaluating an expression
- **Local data** belonging to the procedure
- **Saved machine status** indicates the status of the machine before procedure call (including return address)
- **Access link**
- **Control link** points to the activation of the caller
- Space for return value for efficiency
- Actual parameters used by calling procedure

Activation record

- Each time a procedure is called, space for local variable is pushed onto stack
- When the procedure returns, the space is popped off
- If the procedure p calls procedure q
  - The activation of q may terminate normally. Control resumes from the point from where the call was made
  - If q or some procedure q calls aborts, p ends simultaneously
  - If q throws an exception, p handles if it can or terminates

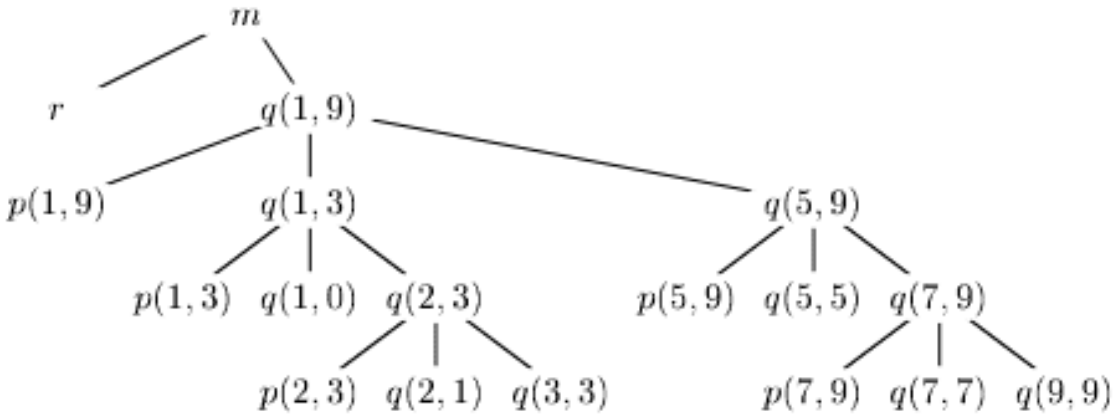
Quicksort program

```
int a[11];
void readArray() { /* Reads 9 integers into a[1], ..., a[9]. */
    int i;
    ...
}
int partition(int m, int n) {
    /* Picks a separator value v, and partitions a[m..n] so that
       a[m..p-1] are less than v, a[p] = v, and a[p+1..n] are
       equal to or greater than v. Returns p. */
    ...
}
void quicksort(int m, int n) {
    int i;
    if (n > m) {
        i = partition(m, n);
        quicksort(m, i-1);
        quicksort(i+1, n);
    }
}
main() {
    readArray();
    a[0] = -9999;
    a[10] = 9999;
    quicksort(1,9);
}
```

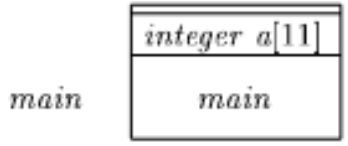
Activation records

```
enter main()
    enter readArray()
    leave readArray()
    enter quicksort(1,9)
        enter partition(1,9)
        leave partition(1,9)
        enter quicksort(1,3)
            ...
        leave quicksort(1,3)
        enter quicksort(5,9)
            ...
        leave quicksort(5,9)
    leave quicksort(1,9)
leave main()
```

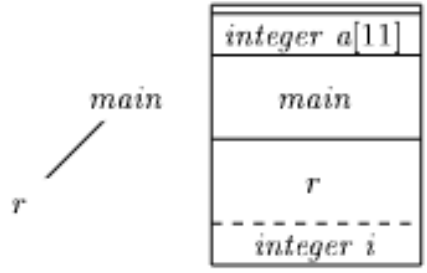
Figure 7.3: Possible activations for the program of Fig. 7.2



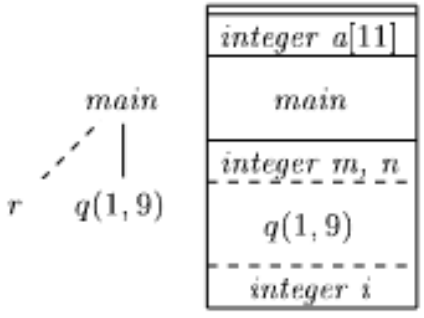
Control stack



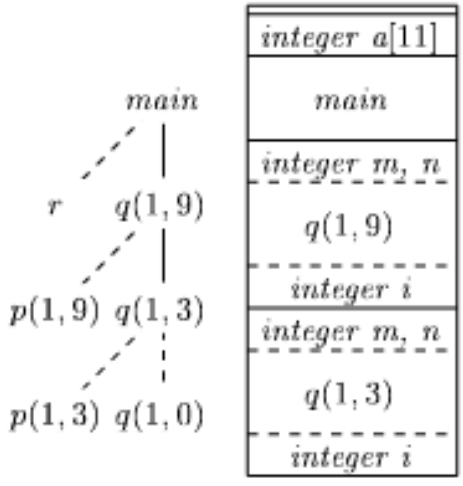
(a) Frame for *main*



(b) r is activated



(c) r has been popped and $q(1,9)$ pushed

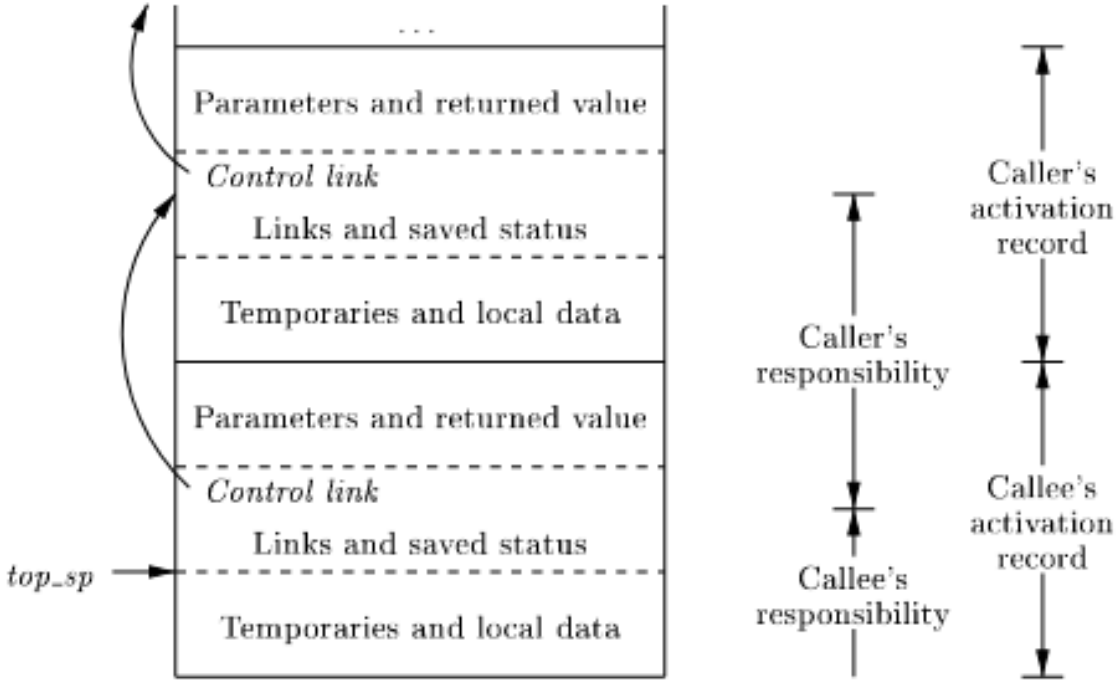


(d) Control returns to $q(1,3)$

Activation tree

- The sequence of procedure calls correspond to a preorder traversal of the activation tree
- The sequence of returns corresponds to a postorder traversal of the activation tree
- It is managed by a run-time stack
- The root stays at the bottom
- The activation record of the currently active procedure stays at the top

Calling sequence



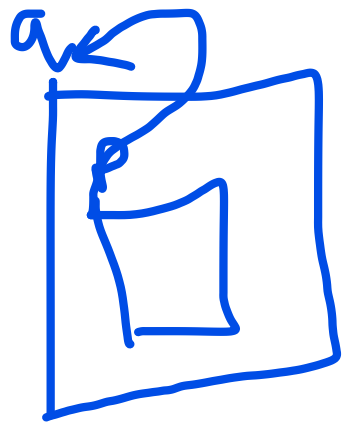
- The top of stack pointer is set to the end of the machine status field of the activation record
- Fixed length data can be reached by fixed length offset

Calling sequence

- Caller evaluates the actual parameters
- It also stores the old value of top pointer to the old value in the callee's control link position. It also stored the return address and set top pointer to end of saved status field
- Callee saves the registers and other status information
- It initializes the local data and begins execution

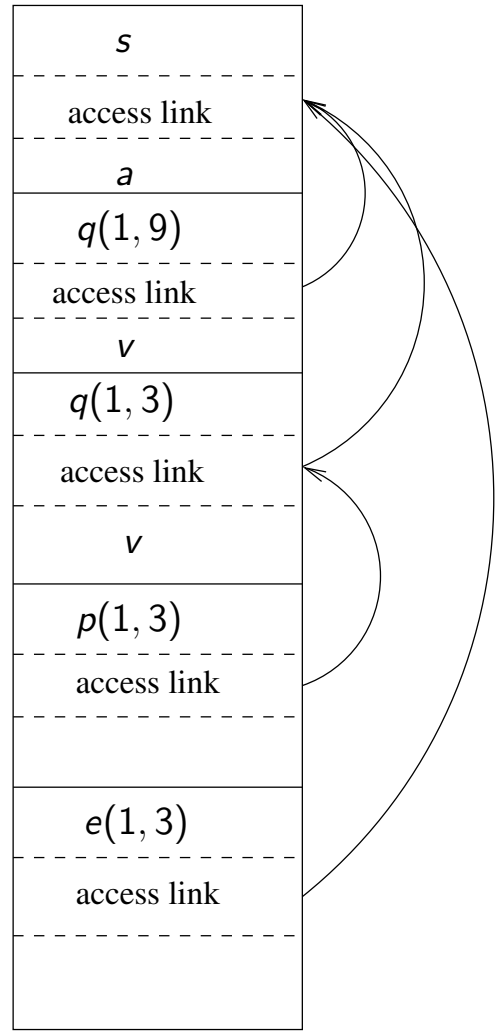
Access link

- If a procedure is nested immediately within procedure q in the source code, then the access link in activation of p points to the most recent activation of q



```
void sort(inputFile, outputFile)
{
  int a[11];
  void readArray(inputFile){···}
  ···
  void exchange(i, j){···}
  ···
  void quicksort(m, n){··· void partition(y, z){
  ··· exchange ···}
  ··· partition ··· quicksort ···}

}
```



Overlapped storage