Lecture No.08

Data Structures

Converting Infix to Postfix

- Example: (A + B) * C

symb	postfix	stack
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB +	
*	AB +	*
C	AB + C	*
	AB + C *	

C++ Templates

- We need a stack of operands and a stack of operators.
- Operands can be integers and floating point numbers, even variables.
- Operators are single characters.
- We would have to create classes FloatStack and CharStack.
- Yet the internal workings of both classes is the same.

C++ Templates

- We can use C++ Templates to create a “template” of a stack class.
- Instantiate float stack, char stack, or stack for any type of element we want.

Stack using templates

Stack.h:

```
template <class T>
class Stack {
public:
    Stack();
    int empty(void); // 1=true, 0=false
    int push(T &); // 1=successful,0=stack overflow
    T pop(void);
    T peek(void);
    ~Stack();
private:
    int top;
    T* nodes;
};
```

Stack using templates

Stack.cpp

```
#include <iostream.h>
#include <stdlib.h>
#include "Stack.cpp"

#define MAXSTACKSIZE 50

template <class T>
Stack<T>::Stack()
{
    top = -1;
    nodes = new T[MAXSTACKSIZE];
}
```

Stack using templates

Stack.cpp

```
template <class T>
Stack<T>::~Stack()
{
    delete nodes;
}

template <class T>
int Stack<T>::empty(void)
{
    if( top < 0 ) return 1;
    return 0;
}
```

Stack using templates

Stack.cpp

```
template <class T>
int Stack<T>::push(T& x)
{
    if( top < MAXSTACKSIZE ) {
        nodes[++top] = x;
        return 1;
    }
    cout << "stack overflow in push.\n";
    return 0;
}
```

Stack using templates

Stack.cpp

```
template <class T>
T Stack<T>::pop(void)
{
    T x;
    if( !empty() ) {
        x = nodes[top--];
        return x;
    }
    cout << "stack underflow in pop.\n";
    return x;
}
```

Stack using templates

main.cpp

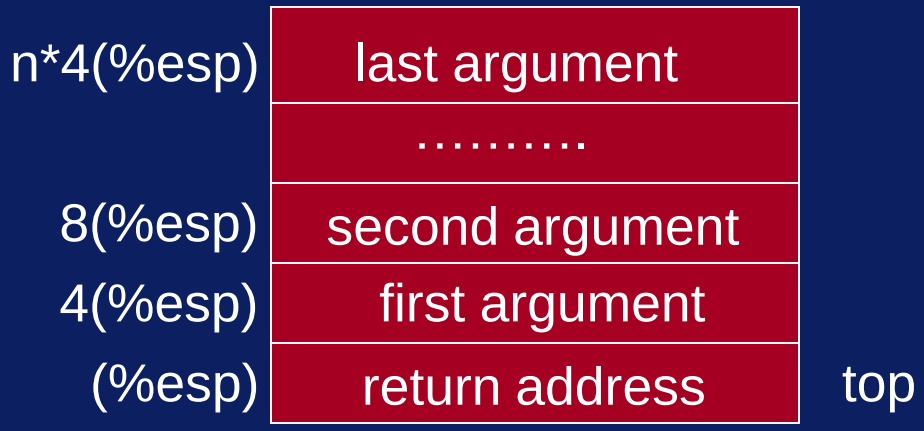
```
#include "Stack.cpp"
int main(int argc, char *argv[]) {
  Stack<int> intstack;
  Stack<char> charstack;
  int x=10, y=20;
  char c='C', d='D';

  intstack.push(x);   intstack.push(y);
  cout << "intstack: " << intstack.pop() << ", "
       << intstack.pop() << "\n";

  charstack.push(c); charstack.push(d);
  cout << "charstack: " << charstack.pop() << ",
  "
       << charstack.pop() << "\n";
}
```

Function Call Stack

- Stacks play a key role in implementation of function calls in programming languages.
- In C++, for example, the “call stack” is used to pass function arguments and receive return values.
- The call stack is also used for “local variables”

Call Stack

- In GCC, a popular C/C++ compiler on Intel platform, stack entries are:



Call Stack

Example: consider the function:

```
int i_avg (int a, int b)
 {
   return (a + b) / 2;
 }

 # Stack layout on entry:
 #
 # 8(%esp) b
 # 4(%esp) a
 #  (%esp) return address
```

Call Stack

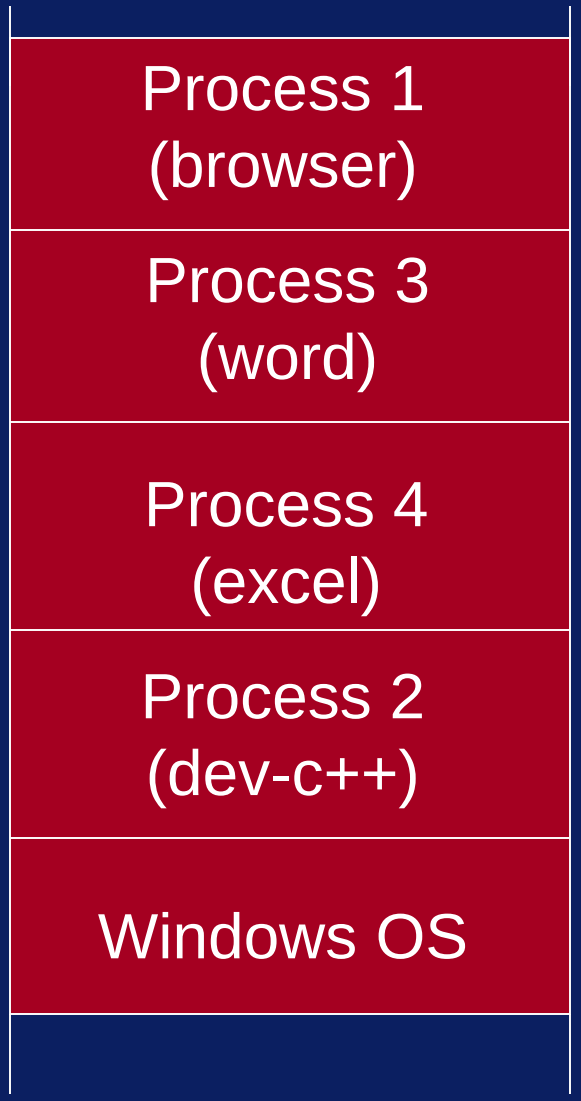
Example: consider the function:

```
int i_avg (int a, int b)
{
  return (a + b) / 2;
}
```

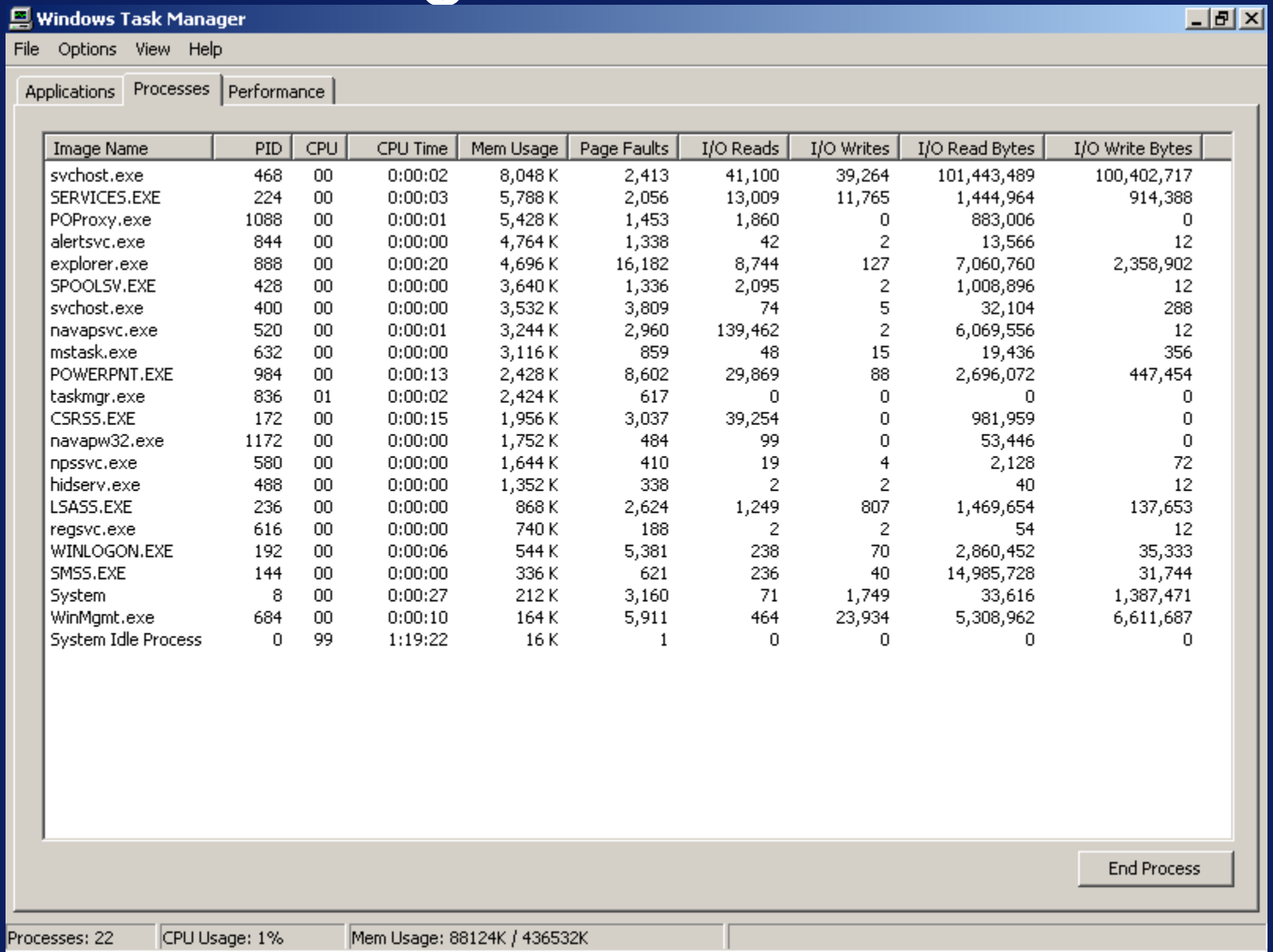
```
.globl _i_avg
_i_avg:
    movl 4(%esp), %eax
    addl 8(%esp), %eax # Add the args
    sarl $1, %eax # Divide by 2
    ret # Return value is in %eax
```

Memory Organization

- When a program (.exe) is run, it is loaded in memory. It becomes a *process*.
- The process is given a block of memory.
- [Control-Alt-DEL]



Task Manager



Memory Organization