

Multitasking



Arijit Mondal

Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
arijit@iitp.ac.in

Introduction

- Midlevel mechanism that are used in software to provide concurrent execution of sequential code
- Concurrent execution is primarily required to improve performance
 - Reduces latency
- Concurrency can come from different aspects
 - Runs on multiprocessor or multicore environment
 - Control the timing of external interaction
- Interrupt - low level multitasking, State machine - high level view

Layers of abstraction

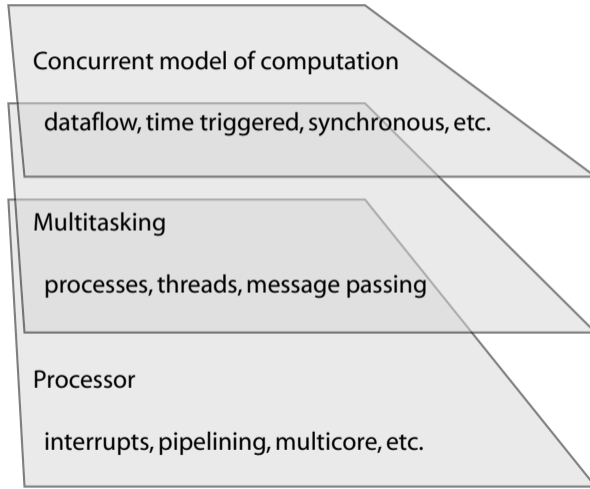


image source: [Introduction to Embedded Systems book](#)

Observer pattern

```
// Value that when updated triggers notification of registered listeners.
int value;

// List of listeners. A linked list containing, pointers to notify procedures.
typedef void* notifyProcedure(int);
struct element {notifyProcedure *listener; struct element* next;};
typedef struct element elementType;
elementType* head = 0;
elementType* tail = 0;

// Procedure to add a listener to the list.
void addListener(notifyProcedure listener) ...

// Procedure to update the value
void update(int newValue) ...

// Procedure to call when notifying
void print(int newValue) ...
```

addListener procedure

```
// Procedure to add a listener.
void addListener(notifyProcedure* listener) {
    if (head == 0) {
        head = malloc(sizeof(element_t));
        head->listener = listener;
        head->next = 0;
        tail = head;
    } else {
        tail->next = malloc(sizeof(element_t));
        tail = tail->next;
        tail->listener = listener;
        tail->next = 0;
    }
}
```

update & print procedure

```
    // Procedure to update x.
28. void update(int newx) {
29.     x = newx;
30.     // Notify listeners.
31.     element_t* element = head;
32.     while (element != 0) {
33.         (*(element->listener))(newx);
34.         element = element->next;
35.     }
36. }
```

```
    // Example of notify procedure.
void print(int arg) {
printf("%d ", arg);
}
```

State diagram

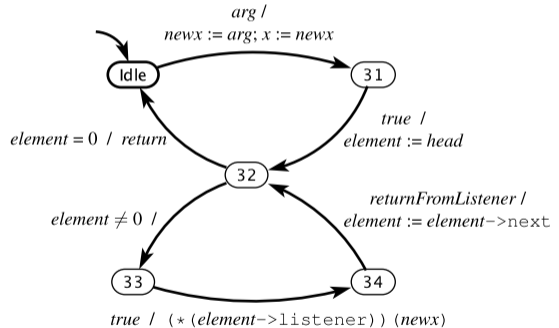
- State machine is determined by variables and their values
- Extended state machine can be used to model program

inputs: *arg*: int, *returnFromListener*: pure

outputs: *return*: pure

local variables: *newx*: int, *element*: element_t*

global variables: *x*: int, *head*: element_t*



Example

```
int main(void) {  
    addListener(&print);  
    addListener(&print);  
    update(1);  
    addListener(&print);  
    update(2);  
    return 0;  
}
```


Threads

- Threads are imperative programs that run concurrently and share memory space
- Operating system provides mechanism in form of collection of procedures which is known as APIs
 - pthreads or POSIX threads
- Each thread has its own stack

Example

```
#include <pthread.h>
#include <stdio.h>
void printN(void *arg){
    int i;
    for (i=0; i<10; i++){
        printf("my ID: %d\n",*(int*)arg);
    }
    return NULL;
}
int main(void){
    pthread_t threadID1, threadID2;
    void exitStatus; int x1=1, x2=2;
    pthread_create(&threadID1,NULL,printN,&x1);
    pthread_create(&threadID2,NULL,printN,&x1);
    printf("Started threads\n");
    pthread_join(threadID1,&exitStatus);
    pthread_join(threadID2,&exitStatus);
}
```

Example

```
pthread_t createThread(int x){  
    pthread_t ID;  
    pthread_create(&ID,NULL,printN,&x);  
    return ID;  
}
```

Implementing threads

- Scheduler decides which thread to execute
 - Equal opportunity to execute (fairness)
 - Timing constraints
 - Importance priority
- Cooperative multitasking
 - Does not interrupt a thread unless the thread itself calls a certain procedure
 - A task can starve
 - Jiffy - time interval at which system clock ISR is invoked
 - Typical values varies between 1ms to 10ms in linux
 - Balancing performance

Race condition

- Two threads are trying to operate on the same variable

counter++	counter--
S11: reg1 = counter	S21: reg2 = counter
S12: reg1 = reg1 + 1	S22: reg2 = reg2 - 1
S13: counter = reg1	S23: counter = reg2

- Outcome for the following sequence if counter starts with a value of 5
 - S11, S12, S21, S22, S13, S23
- To prevent race condition use **mutual exclusion lock (mutex)**

addListener procedure

```
pthread_mutex_t lock=PTHREAD_MUTEX_INITIALIZER;
void addListener(notifyProcedure* listener) {
    pthread_mutex_lock(&lock);
    if (head == 0) {
        head = malloc(sizeof(element_t));
        head->listener = listener;
        head->next = 0;
        tail = head;
    } else {
        tail->next = malloc(sizeof(element_t));
        tail = tail->next;
        tail->listener = listener;
        tail->next = 0;
    }
    pthread_mutex_unlock(&lock);
}
```

Deadlock

- Suppose threads A and B needs two locks lock1 and lock2 to enter critical section
- Suppose A acquires lock1 and B acquires lock2
 - No progress