History

We are now going to look at some history: this is useful as it will illustrate the many important parts of OSs and why they are necessary

History

We are now going to look at some history: this is useful as it will illustrate the many important parts of OSs and why they are necessary

At first, computers had no operating systems (1960s)

History

We are now going to look at some history: this is useful as it will illustrate the many important parts of OSs and why they are necessary

At first, computers had no operating systems (1960s)

- Every programmer had to write their programs for the particular machine they were using

History

We are now going to look at some history: this is useful as it will illustrate the many important parts of OSs and why they are necessary

At first, computers had no operating systems (1960s)

- Every programmer had to write their programs for the particular machine they were using
- So no portability

History

We are now going to look at some history: this is useful as it will illustrate the many important parts of OSs and why they are necessary

At first, computers had no operating systems (1960s)

- Every programmer had to write their programs for the particular machine they were using
- So no portability
- And lots of repeated code between programs (“write a character to the teletype”)

History

We are now going to look at some history: this is useful as it will illustrate the many important parts of OSs and why they are necessary

At first, computers had no operating systems (1960s)

- Every programmer had to write their programs for the particular machine they were using
- So no portability
- And lots of repeated code between programs (“write a character to the teletype”)

Remember: the more we make programmers do, the more likely they are going to make a mistake

History

Furthermore, programmers rarely even saw the computer

History

Furthermore, programmers rarely even saw the computer

- The program and the data is needed (collectively called a *job*) would be prepared on paper tape or punched card

History

Furthermore, programmers rarely even saw the computer

- The program and the data is needed (collectively called a *job*) would be prepared on paper tape or punched card
- Jobs would be given to operators who would load and run them, collect the results and then send the results back to the programmer

History

Furthermore, programmers rarely even saw the computer

- The program and the data is needed (collectively called a *job*) would be prepared on paper tape or punched card
- Jobs would be given to operators who would load and run them, collect the results and then send the results back to the programmer
- Usually, there would be a bug and the programmer would have to fix the program and go round the loop again

History

Furthermore, programmers rarely even saw the computer

- The program and the data is needed (collectively called a *job*) would be prepared on paper tape or punched card
- Jobs would be given to operators who would load and run them, collect the results and then send the results back to the programmer
- Usually, there would be a bug and the programmer would have to fix the program and go round the loop again
- As computer time was limited, there was an issue of *scheduling* jobs, initially done by hand

History

Furthermore, programmers rarely even saw the computer

- The program and the data is needed (collectively called a *job*) would be prepared on paper tape or punched card
- Jobs would be given to operators who would load and run them, collect the results and then send the results back to the programmer
- Usually, there would be a bug and the programmer would have to fix the program and go round the loop again
- As computer time was limited, there was an issue of *scheduling* jobs, initially done by hand
- Turnaround on jobs could be days

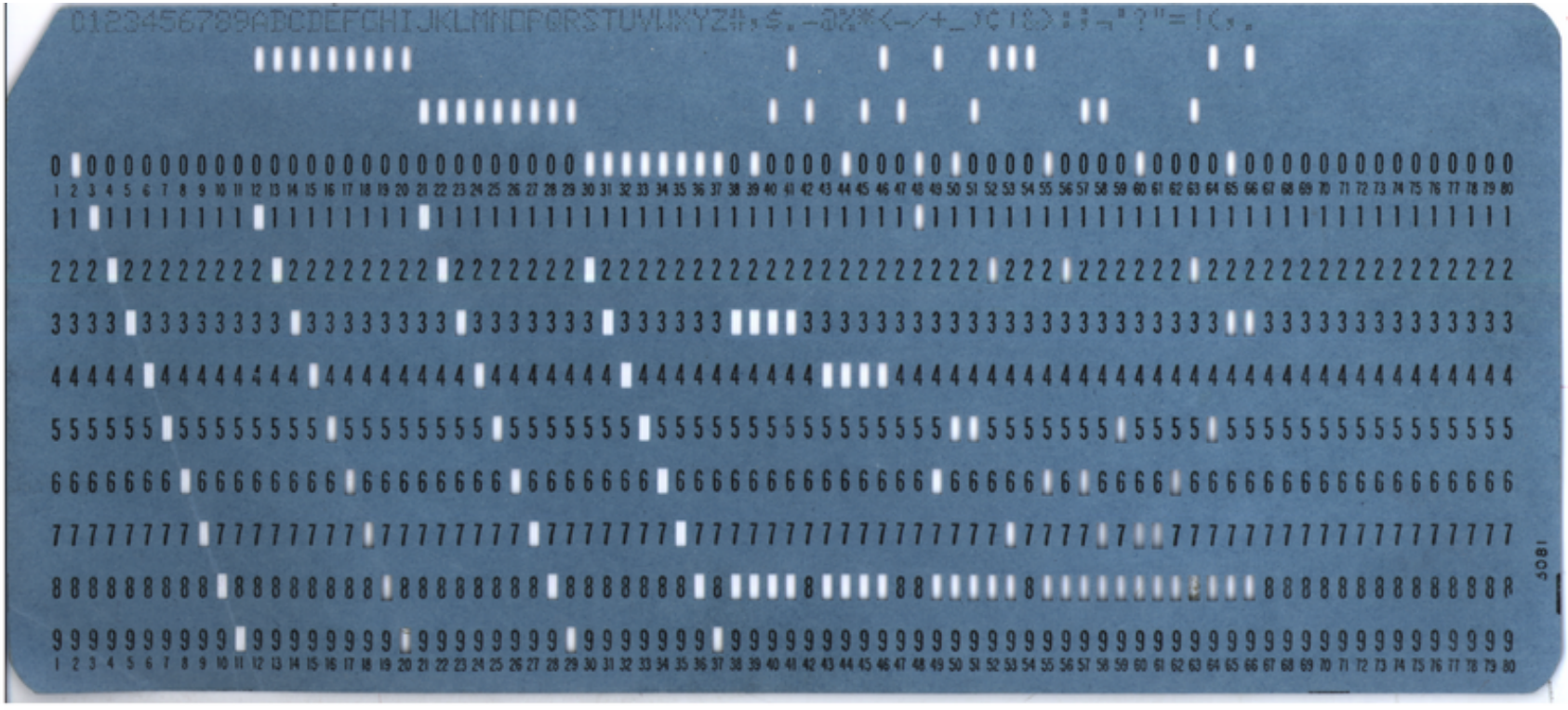
History

Furthermore, programmers rarely even saw the computer

- The program and the data is needed (collectively called a *job*) would be prepared on paper tape or punched card
- Jobs would be given to operators who would load and run them, collect the results and then send the results back to the programmer
- Usually, there would be a bug and the programmer would have to fix the program and go round the loop again
- As computer time was limited, there was an issue of *scheduling* jobs, initially done by hand
- Turnaround on jobs could be days

This concentrated the programmer's mind wonderfully!

History



From Wikipedia. Encodes a single 80 character line

History

From Wikipedia. 5 and 8 hole paper tapes

History

It was soon found there was a lot of repeated code between programs, so useful tools (programs and libraries of code) were developed to help manage repetitive tasks

History

It was soon found there was a lot of repeated code between programs, so useful tools (programs and libraries of code) were developed to help manage repetitive tasks

- collecting common functions in *system libraries* (sqrt, open file, etc.)

History

It was soon found there was a lot of repeated code between programs, so useful tools (programs and libraries of code) were developed to help manage repetitive tasks

- collecting common functions in *system libraries* (sqrt, open file, etc.)
- program management (loaders)

History

It was soon found there was a lot of repeated code between programs, so useful tools (programs and libraries of code) were developed to help manage repetitive tasks

- collecting common functions in *system libraries* (sqrt, open file, etc.)
- program management (loaders)
- debuggers

History

It was soon found there was a lot of repeated code between programs, so useful tools (programs and libraries of code) were developed to help manage repetitive tasks

- collecting common functions in *system libraries* (sqrt, open file, etc.)
- program management (loaders)
- debuggers
- Interfacing to hardware: I/O drivers (send file to printer, etc.)

History

It was soon found there was a lot of repeated code between programs, so useful tools (programs and libraries of code) were developed to help manage repetitive tasks

- collecting common functions in *system libraries* (sqrt, open file, etc.)
- program management (loaders)
- debuggers
- Interfacing to hardware: I/O drivers (send file to printer, etc.)

This made programming and program management easier, but there was still lots of human intervention needed

History

The issue was to keep the big and expensive computer as busy as possible, running programs all the time

History

The issue was to keep the big and expensive computer as busy as possible, running programs all the time

Idle time was a waste of money

History

The issue was to keep the big and expensive computer as busy as possible, running programs all the time

Idle time was a waste of money

So the operators would load many programs on to a fast medium, such as magnetic tape, and the computer would load and run them as fast as hardware allowed

History

The issue was to keep the big and expensive computer as busy as possible, running programs all the time

Idle time was a waste of money

So the operators would load many programs on to a fast medium, such as magnetic tape, and the computer would load and run them as fast as hardware allowed

This was called *spooling*, the first instance of addressing the disparity between human and computer speeds

History

Spooling would also be used on output: the output would be written to a mag tape, which could then later be attached to a printer

History

Spooling would also be used on output: the output would be written to a mag tape, which could then later be attached to a printer

Again, this was because printers are slower than computers

History

Of course, this was soon automated: have a little program, called a *monitor* (or *supervisor*), that loads programs from tape; runs them; and puts the results on tape

History

Of course, this was soon automated: have a little program, called a *monitor* (or *supervisor*), that loads programs from tape; runs them; and puts the results on tape

This would be directed by a *job control language*

History

A famous job control language from IBM was called JCL

History

A famous job control language from IBM was called JCL

Of course “JCL” means “Job Control Language”, but JCL was just one of a few job control languages

History

```
//IS198CPY JOB (IS198T30500),'COPY JOB',CLASS=L,MSGCLASS=X
//COPY01   EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DSN=OLDFILE,DISP=SHR
//SYSUT2   DD  DSN=NEWFILE,
//           DISP=(NEW,CATLG,DELETE),
//           SPACE=(CYL,(40,5),RLSE),
//           DCB=(LRECL=115,BLKSIZE=1150)
//SYSIN    DD  DUMMY
```

(From Wikipedia) Any guesses?

History

```
//IS198CPY JOB (IS198T30500),'COPY JOB',CLASS=L,MSGCLASS=X
//COPY01   EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DSN=OLDFILE,DISP=SHR
//SYSUT2   DD  DSN=NEWFILE,
//           DISP=(NEW,CATLG,DELETE),
//           SPACE=(CYL,(40,5),RLSE),
//           DCB=(LRECL=115,BLKSIZE=1150)
//SYSIN    DD  DUMMY
```

(From Wikipedia) Any guesses?

This copies OLDFILE to NEWFILE

History

```
//IS198CPY JOB (IS198T30500),'COPY JOB',CLASS=L,MSGCLASS=X
//COPY01   EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DSN=OLDFILE,DISP=SHR
//SYSUT2   DD  DSN=NEWFILE,
//           DISP=(NEW,CATLG,DELETE),
//           SPACE=(CYL,(40,5),RLSE),
//           DCB=(LRECL=115,BLKSIZE=1150)
//SYSIN    DD  DUMMY
```

(From Wikipedia) Any guesses?

This copies OLDFILE to NEWFILE
This would be set on 9 punched cards

History

A Fortran program, with data:

```
//CONVERT JOB USER=UGA001,MSGCLASS=6,NOTIFY=UGA001
//*MAIN CLASS=NITE,LINES=40,ORG=UGAIBM1.LOCAL
// EXEC FORTVCLG,REGION=2000K
//FORT.SYSIN DD *
       READ(5,10) CENT
    10 FORMAT(F6.2)
       FAHR=(CENT*9.0/5.0)+32.0
       WRITE(6,20) CENT,FAHR
    20 FORMAT(F6.2,' CENT = ',F6,2,'FAHR')
       STOP
       END
/*
//GO.SYSIN DD *
100.00
/*
//
```

History

JCL allowed several programs to be collected and loaded together in a single bunch

History

JCL allowed several programs to be collected and loaded together in a single bunch

This is called *batch processing*

History

JCL allowed several programs to be collected and loaded together in a single bunch

This is called *batch processing*

Running in batches is more efficient, as we spend more time running our programs and less time messing around in the overheads of loading and unloading

History

This might seem like ancient history, but these things are still happening

History

This might seem like ancient history, but these things are still happening

Modern large computers are managed in just this way: and for the same reasons

History

This might seem like ancient history, but these things are still happening

Modern large computers are managed in just this way: and for the same reasons

We still run jobs and need to manage CPU usage and disk usage

History

This might seem like ancient history, but these things are still happening

Modern large computers are managed in just this way: and for the same reasons

We still run jobs and need to manage CPU usage and disk usage

Turnaround is seconds or minutes rather than days, but the principle is the same

History

This might seem like ancient history, but these things are still happening

Modern large computers are managed in just this way: and for the same reasons

We still run jobs and need to manage CPU usage and disk usage

Turnaround is seconds or minutes rather than days, but the principle is the same

Exercise look up *Portable Batch System*, PBS and compare with JCL

History

So the monitor was just a program there to help manage the machine

History

So the monitor was just a program there to help manage the machine

It would load an application into memory (from tape or wherever)

History

So the monitor was just a program there to help manage the machine

It would load an application into memory (from tape or wherever)

And then jump to the start of the application and start executing it

History

So the monitor was just a program there to help manage the machine

It would load an application into memory (from tape or wherever)

And then jump to the start of the application and start executing it

When the application finished, it would (be expected to) jump back to the monitor, so the monitor could deal with the next program

History

But if the application was badly written, it could overwrite the monitor

Machine memory

History

But if the application was badly written, it could overwrite the monitor

Machine memory

Either accidentally or deliberately

History

But if the application was badly written, it could overwrite the monitor

Machine memory

Either accidentally or deliberately

They needed to do something about this

History

But if the application was badly written, it could overwrite the monitor

Machine memory

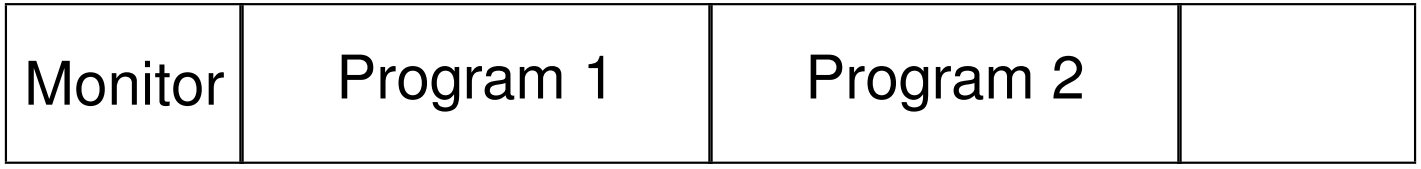
Either accidentally or deliberately

They needed to do something about this

But there are other problems, too

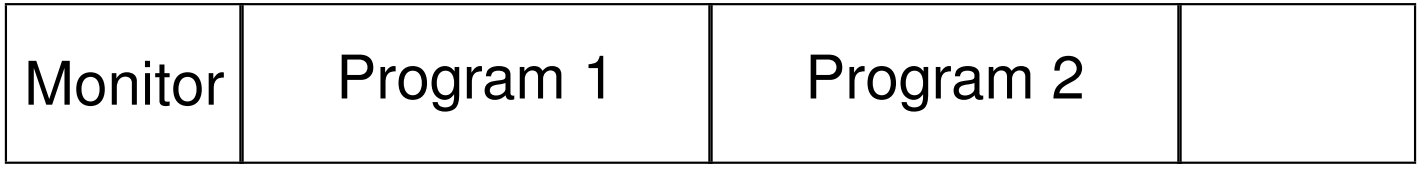
History

It was soon found to be more efficient to load more than one program into memory (when there was space)



History

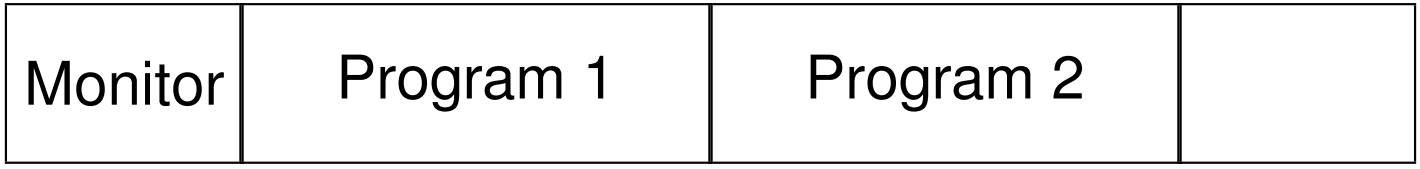
It was soon found to be more efficient to load more than one program into memory (when there was space)



The advantage being that if Program 1 was doing something like writing to a tape that takes a lot of time, but no CPU, the computer could run Program 2 in the meanwhile

History

It was soon found to be more efficient to load more than one program into memory (when there was space)

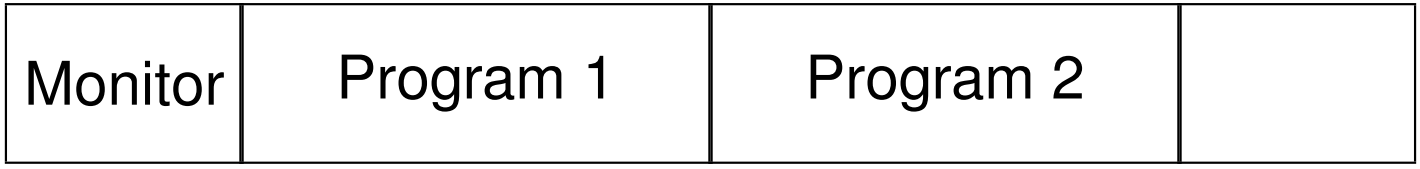


The advantage being that if Program 1 was doing something like writing to a tape that takes a lot of time, but no CPU, the computer could run Program 2 in the meanwhile

When Program 2 pauses and Program 1 needs to run again, the computer could switch back to it

History

It was soon found to be more efficient to load more than one program into memory (when there was space)



The advantage being that if Program 1 was doing something like writing to a tape that takes a lot of time, but no CPU, the computer could run Program 2 in the meanwhile

When Program 2 pauses and Program 1 needs to run again, the computer could switch back to it

The decisions on what to run and actually doing the switching between programs was the job of the monitor

History

Now Program 1 could corrupt Program 2 as well as the monitor!

History

Now Program 1 could corrupt Program 2 as well as the monitor!

Or Program 1 could read confidential data out of Program 2

History

Now Program 1 could corrupt Program 2 as well as the monitor!

Or Program 1 could read confidential data out of Program 2

Some sort of protection of the monitor and other programs is needed

History

Now Program 1 could corrupt Program 2 as well as the monitor!

Or Program 1 could read confidential data out of Program 2

Some sort of protection of the monitor and other programs is needed

What happens if Program 1 goes into an infinite loop?

History

Now Program 1 could corrupt Program 2 as well as the monitor!

Or Program 1 could read confidential data out of Program 2

Some sort of protection of the monitor and other programs is needed

What happens if Program 1 goes into an infinite loop?

Control never returns to the monitor and Program 2 never gets to run

History

Now Program 1 could corrupt Program 2 as well as the monitor!

Or Program 1 could read confidential data out of Program 2

Some sort of protection of the monitor and other programs is needed

What happens if Program 1 goes into an infinite loop?

Control never returns to the monitor and Program 2 never gets to run

Some means of curtailing runaway programs is needed

History

So the “monitor runs the programs”: what does this really mean?

History

So the “monitor runs the programs”: what does this really mean?

Nothing sophisticated. The monitor code just jumps to the program code so the machine is now running the program

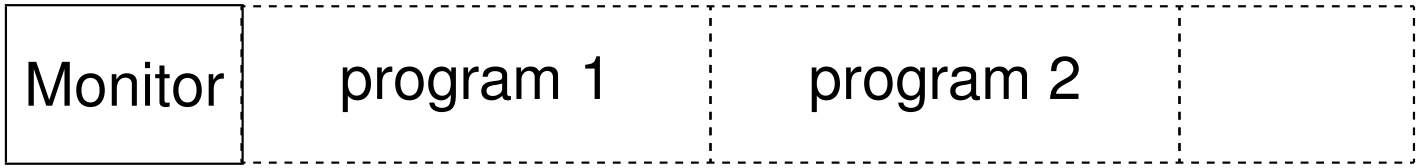
History

So the “monitor runs the programs”: what does this really mean?

Nothing sophisticated. The monitor code just jumps to the program code so the machine is now running the program

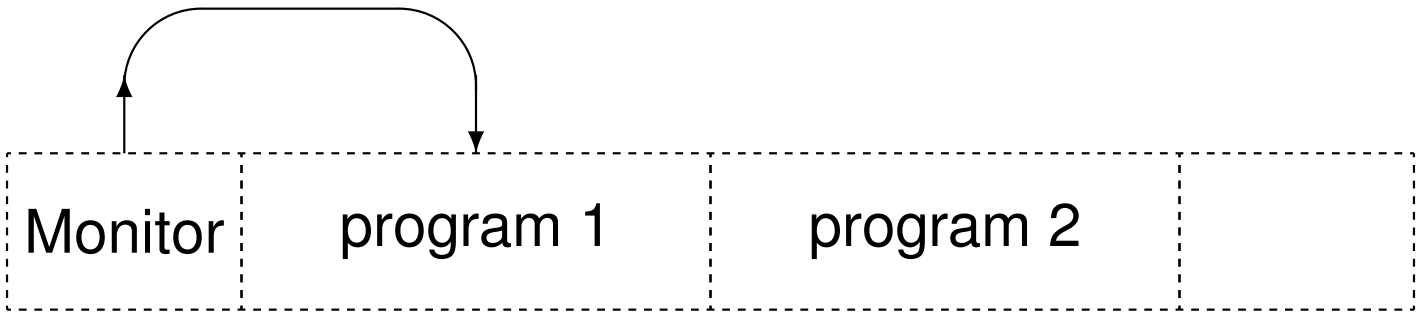
Take care over this point: the monitor doesn’t sit and watch the program running, the monitor is *not* running while the program is running

History



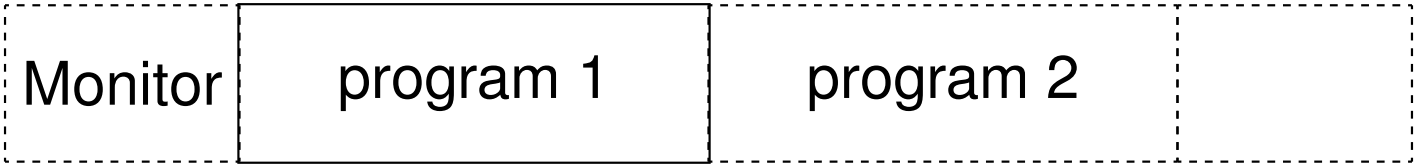
Monitor runs

History



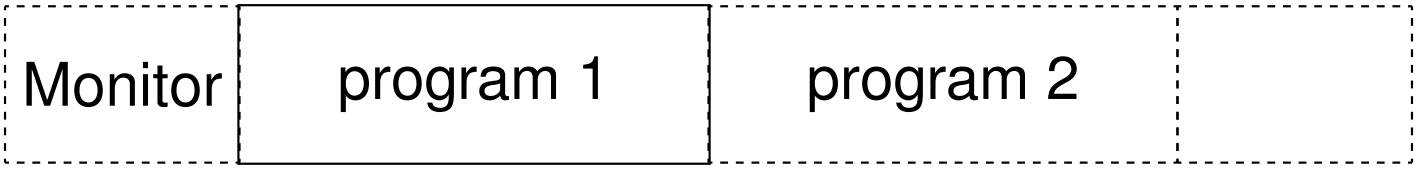
Monitor jumps to program 1

History



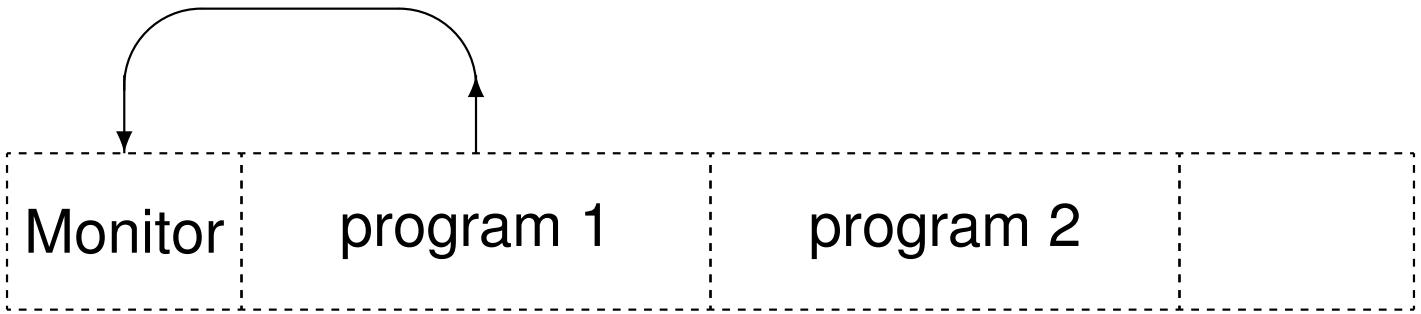
Program runs

History



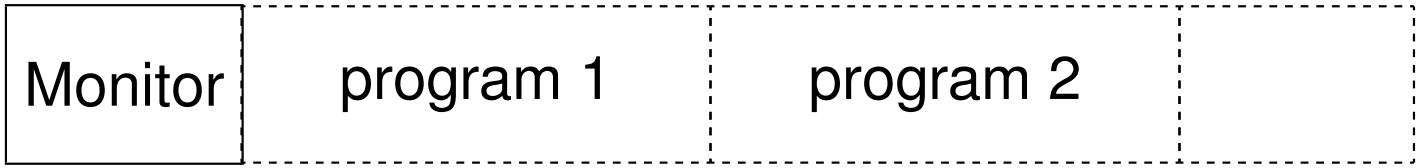
Tape needed

History



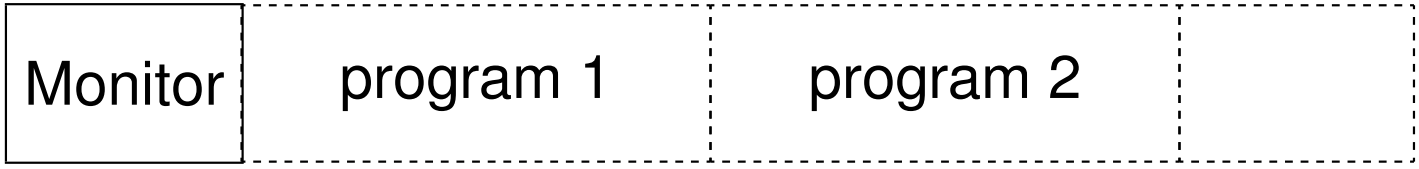
Program calls monitor

History



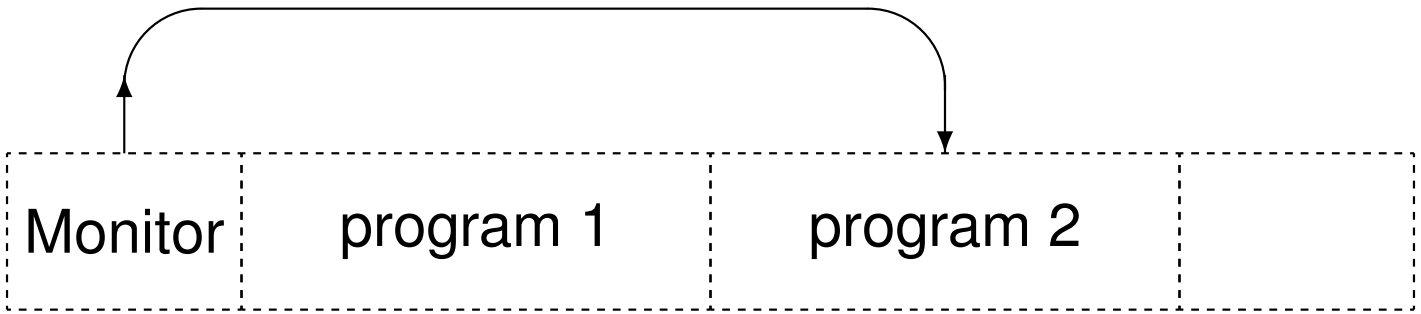
Monitor sets up tape

History



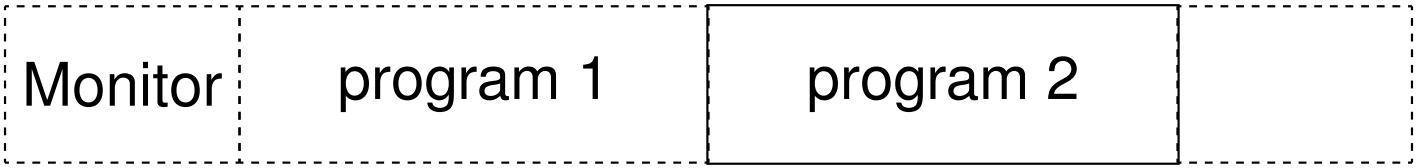
Monitor decides to run another program while waiting for the tape

History



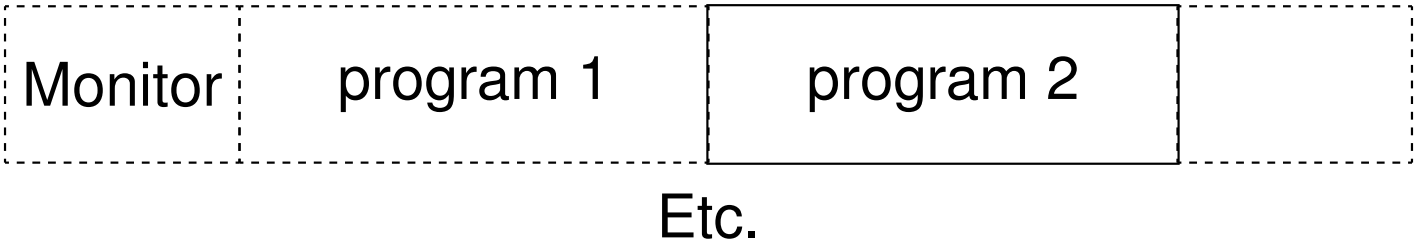
Monitor jumps to program 2

History



Program 2 runs

History



History

There is a *single stream of control* jumping between monitor and one or more programs

History

There is a *single stream of control* jumping between monitor and one or more programs

The monitor is not running when a user program is running, and the user program is not running while the monitor is running

History

This changing between multiple user programs is called *multitasking*. But only one thing is ever running

History

This changing between multiple user programs is called *multitasking*. But only one thing is ever running

Multitasking improves the efficiency of use of a computer since while one program waits for a slow peripheral another program can run

History

So the monitor now needs to keep making decisions on which program to run next: it is *scheduling* the programs

History

So the monitor now needs to keep making decisions on which program to run next: it is *scheduling* the programs

The choices can be made according to many criteria

History

So the monitor now needs to keep making decisions on which program to run next: it is *scheduling* the programs

The choices can be made according to many criteria

- how long a program has been running

History

So the monitor now needs to keep making decisions on which program to run next: it is *scheduling* the programs

The choices can be made according to many criteria

- how long a program has been running
- a *priority* of a program

History

So the monitor now needs to keep making decisions on which program to run next: it is *scheduling* the programs

The choices can be made according to many criteria

- how long a program has been running
- a *priority* of a program
- whether a program is likely to need CPU very soon, or can wait

History

So the monitor now needs to keep making decisions on which program to run next: it is *scheduling* the programs

The choices can be made according to many criteria

- how long a program has been running
- a *priority* of a program
- whether a program is likely to need CPU very soon, or can wait
- how much the owner of the program has paid

History

So the monitor now needs to keep making decisions on which program to run next: it is *scheduling* the programs

The choices can be made according to many criteria

- how long a program has been running
- a *priority* of a program
- whether a program is likely to need CPU very soon, or can wait
- how much the owner of the program has paid
- And many more things

History

Early scheduling algorithms were very simple, e.g., keep running the same program until it's done; later algorithms tried to be more clever

History

Early scheduling algorithms were very simple, e.g., keep running the same program until it's done; later algorithms tried to be more clever

Some programmers would write their programs to take advantage of deficiencies in the scheduling algorithm: in the worst case *starve* other programs of any CPU time at all!

History

Early scheduling algorithms were very simple, e.g., keep running the same program until it's done; later algorithms tried to be more clever

Some programmers would write their programs to take advantage of deficiencies in the scheduling algorithm: in the worst case *starve* other programs of any CPU time at all!

It is tempting to make the scheduling algorithm complicated: but remember more time spent in the monitor deciding what to schedule next is less time for the programs

History

Early scheduling algorithms were very simple, e.g., keep running the same program until it's done; later algorithms tried to be more clever

Some programmers would write their programs to take advantage of deficiencies in the scheduling algorithm: in the worst case *starve* other programs of any CPU time at all!

It is tempting to make the scheduling algorithm complicated: but remember more time spent in the monitor deciding what to schedule next is less time for the programs

So there is a trade-off of making scheduling *fast* but *fair*

History

Early scheduling algorithms were very simple, e.g., keep running the same program until it's done; later algorithms tried to be more clever

Some programmers would write their programs to take advantage of deficiencies in the scheduling algorithm: in the worst case *starve* other programs of any CPU time at all!

It is tempting to make the scheduling algorithm complicated: but remember more time spent in the monitor deciding what to schedule next is less time for the programs

So there is a trade-off of making scheduling *fast* but *fair*

This is still an issue today: we'll look a little into scheduling later