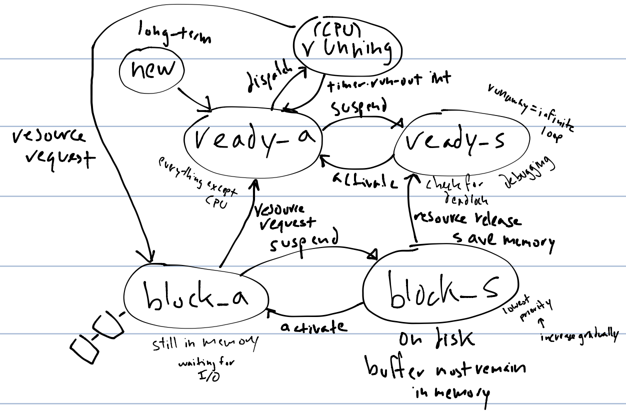
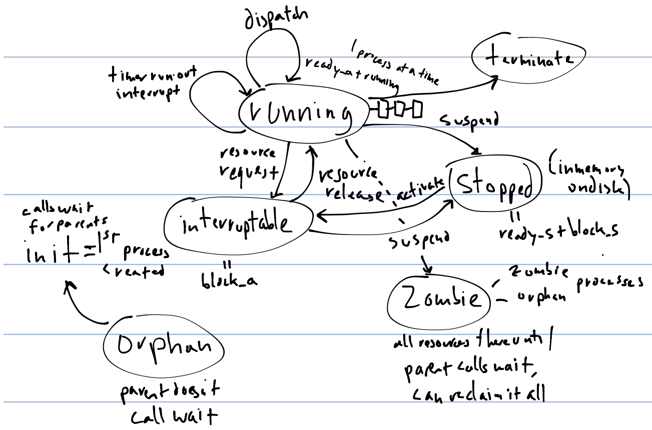
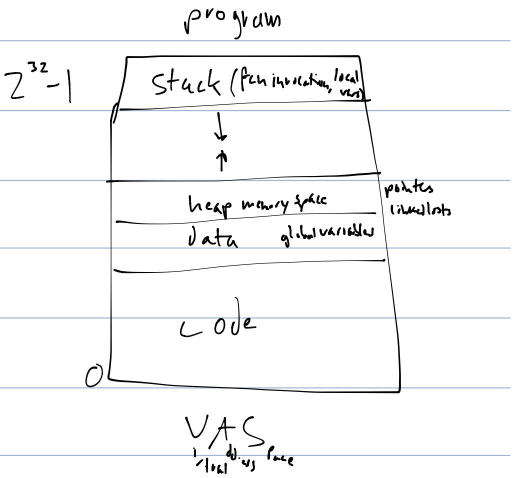


Linux approach





process context switch (when CPU becomes available)

multiprogramming

save PCB for preempt-to-be (old) process (dispatcher)

restore PCB for dispatch-to-be process (dispatcher + scheduler)

context:

(1) uniqueID of the process (pid)

(2) pointer to parent process

(3) pointer to child processes

(4) priority of the process (for CPU scheduling)

(5) a register save area (are used to save register contents) shared by all processes

(6) the processor it's running

(7) list of open files

(8) current position of stack pointer

(9) current position of heap pointer

(10) housekeeping in function

passive termination

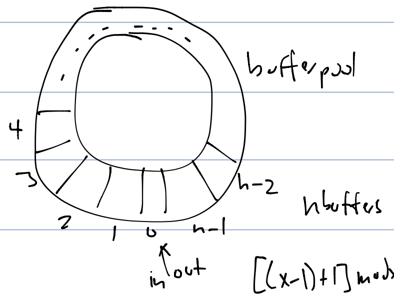
```
    exit();
```

active termination

IPC

(1) approach 1: shared memory

producer-consumer problem



```
struct{
    ....
} item;

const in BUFFER_SIZE = 10;

item buffer[BUFFER_SIZE]

item next_item_produced;,next_item_consumed
```

$[(x-1)+1] \bmod x = 0$

the producer process:

```
while(true){
    while(((int 1)%BUFFER_SIZE)== Out);
    buffer[in]=next_item_produced;
    in=(int1) mod BUFFER_SIZE;
    }
```

the consume process: uses in and out to check if empty

```
    While (true){
        While ( in==out);
        next_item_consumed=buffer[out];
        out =( out+1)% BUFFER_SIZE
        /*consume the item*/
    }
```

producer process:

```
    While (true) {
      /*produce an item in "item_message"*/
      send(consumer, item_message);
    }
```

Consumer process:

```
   while (true){
        receive(producer, item_message)
         /*consumes the item in "item_message"*/
   }
```