MP5 – Overview

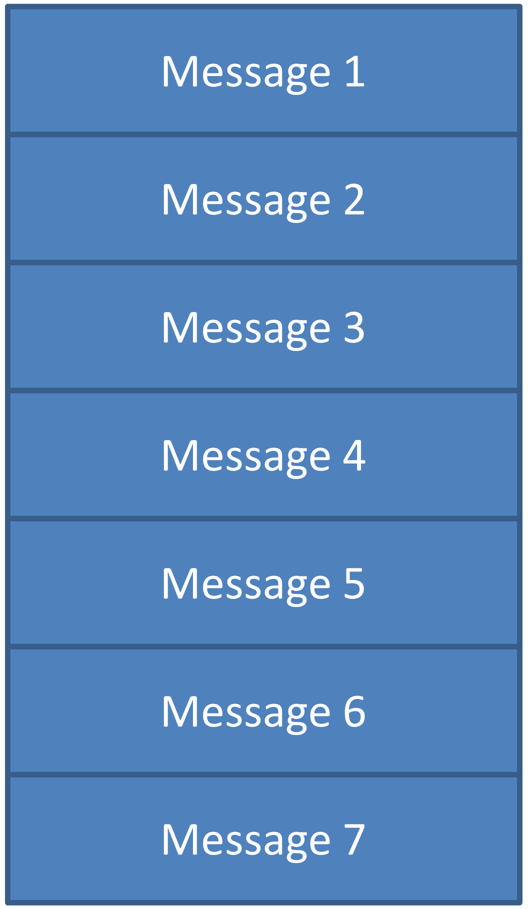
The Assignment

- In this Mp you will be working on several IPC techniques
- The final goal is to implement a multi-process statistics collection tool
- Multiple files can be analyzed CONCURRENTLY
- Each file is analyzed by multiple processes

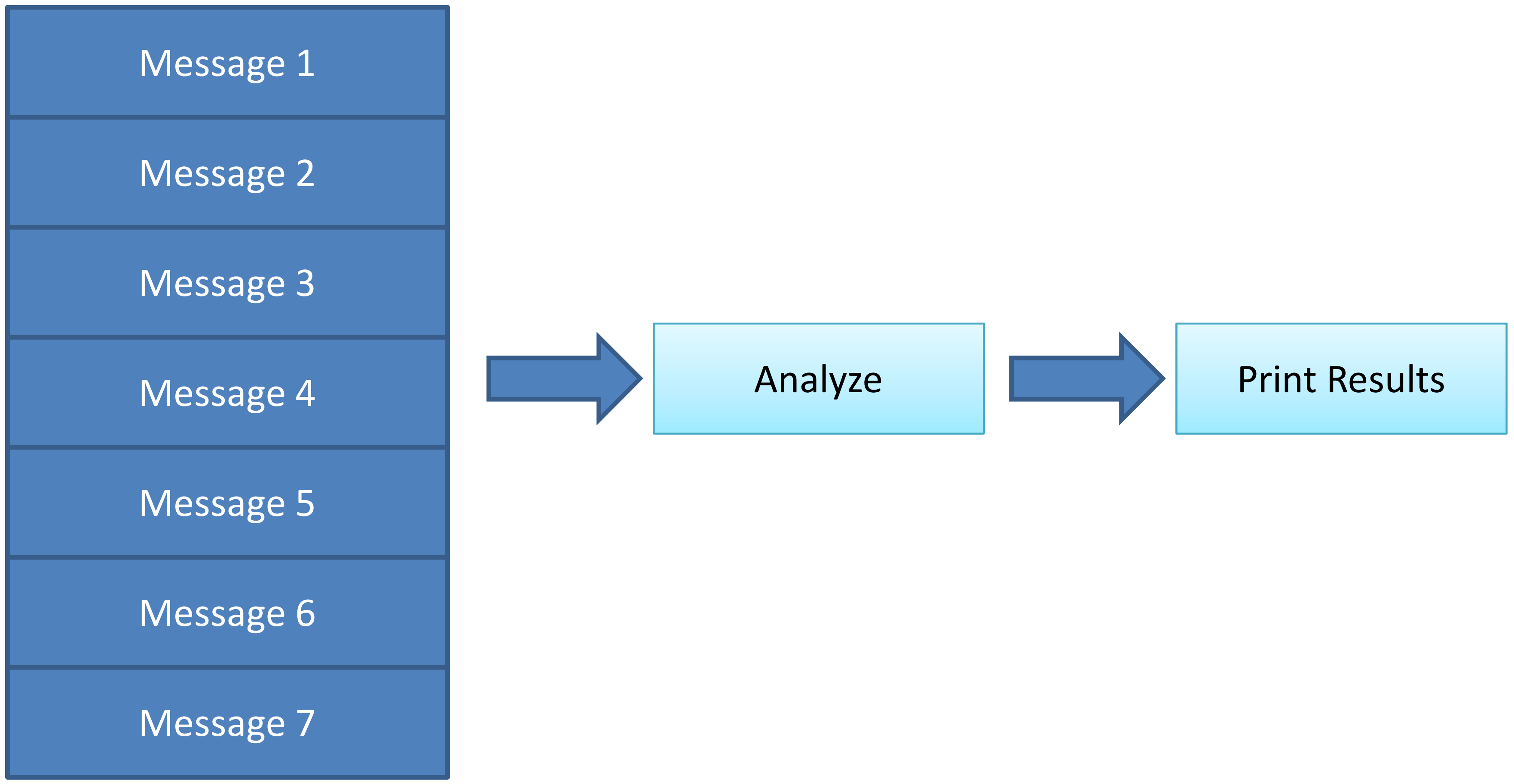
Map - Reduce

- An algorithm used in many contexts (Google!) for analyzing huge datasets in few instants
- Based on a simple divide-and-conquer approach
- Chunks are analyzed by parallel processes

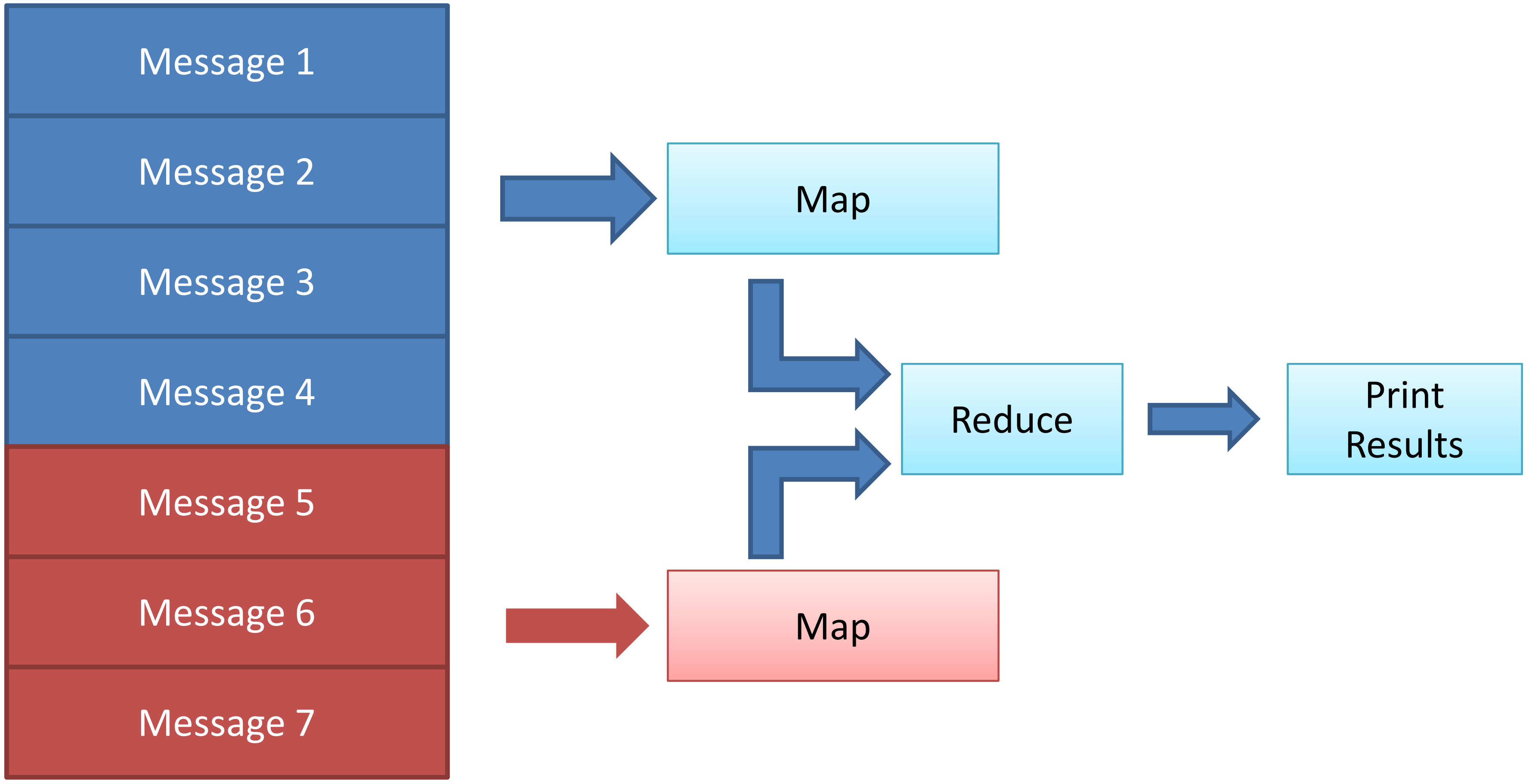
Closer Look at The Assignment



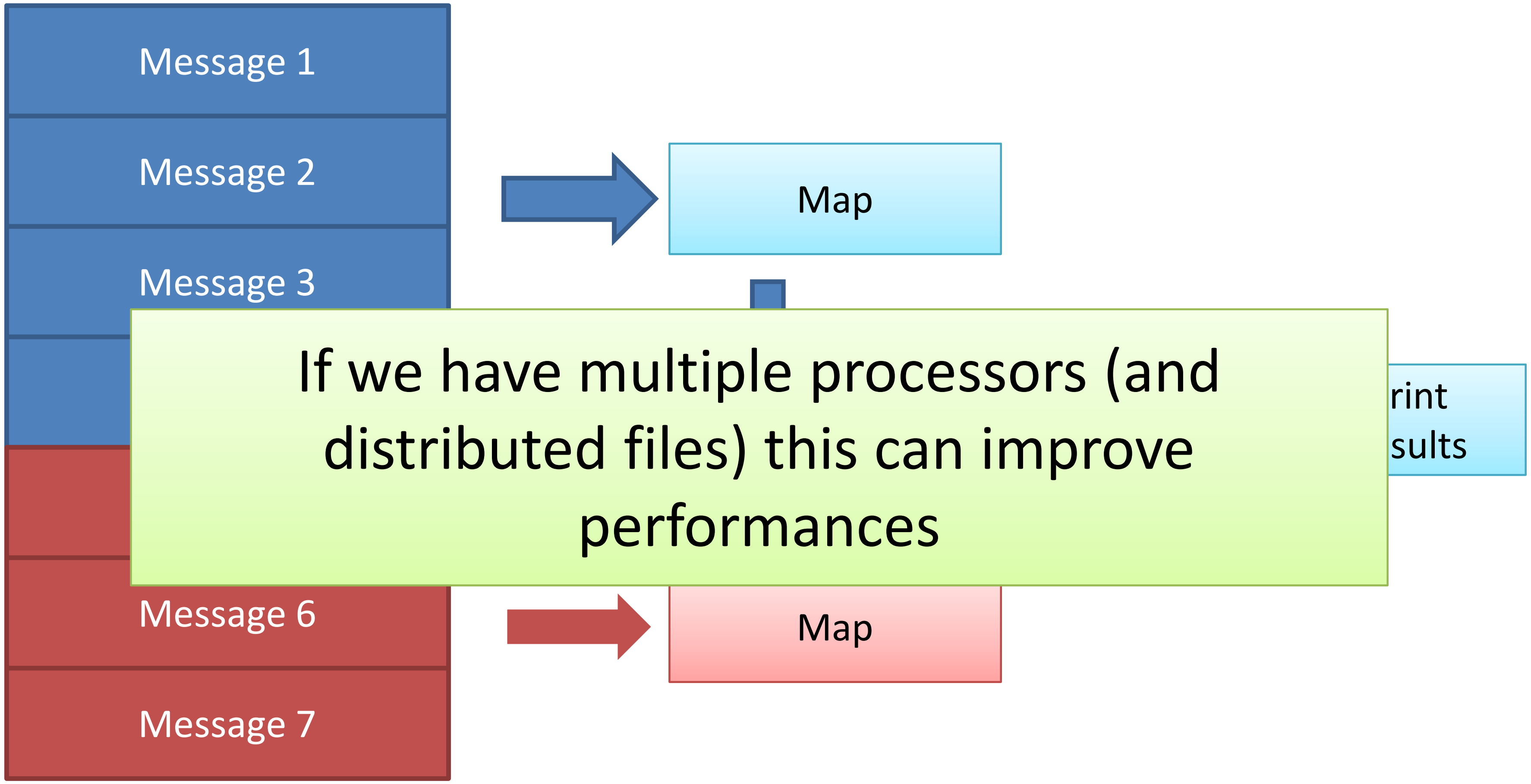
Closer Look at The Assignment



Closer Look at The Assignment



Closer Look at The Assignment

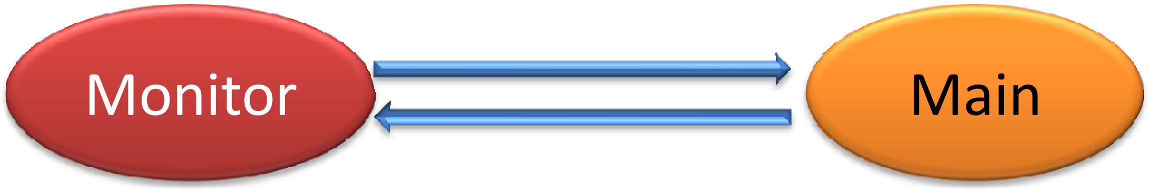


If we have multiple processors (and distributed files) this can improve performances

Why IPC?

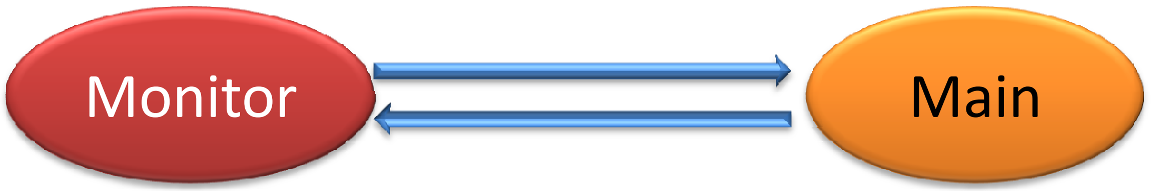
- Memory of different processes is independent
- Need for a way of sharing information among different processes
- Need for synchronization among processes

The Processes Hierarchy



- ✓Two-way handshake using SIGNALS
- ✓Shared Memory:
  - ✓Semaphores
  - ✓Filename to be analyzed
  - ✓Pipe name
- ✓Named Pipe
  - ✓Monitor reads
  - ✓Main writes the results

Two-Way Handshake



1) Start MAIN and get its PID
2) Create a shared memory segment
3) Wait for a Signal
4) Attach the shared memory segment
5) Initializes the Semaphores
6) Write MONITOR PID in shared mem
7) Send USR1 to MAIN
8) Wait for a USR1 Signal
9) Prepare and open a named pipe
10) Save the filename to shmem
11) Send USR1 to MONITOR
12) Open the pipe for reading

1) Start MAIN and get its PID
2) Create a shared memory segment
3) Wait for a USR1 Signal
4) Attach the shared memory segment
5) Initializes the Semaphores
6) Write MONITOR PID in shared mem
7) Send USR1 to MAIN
8) Wait for a USR1 Signal
9) Prepare and open a named pipe
10) Save the filename to shmem
11) Send USR1 to MONITOR
12) Open the pipe for reading

Monitor Controls Main

Monitor
Main

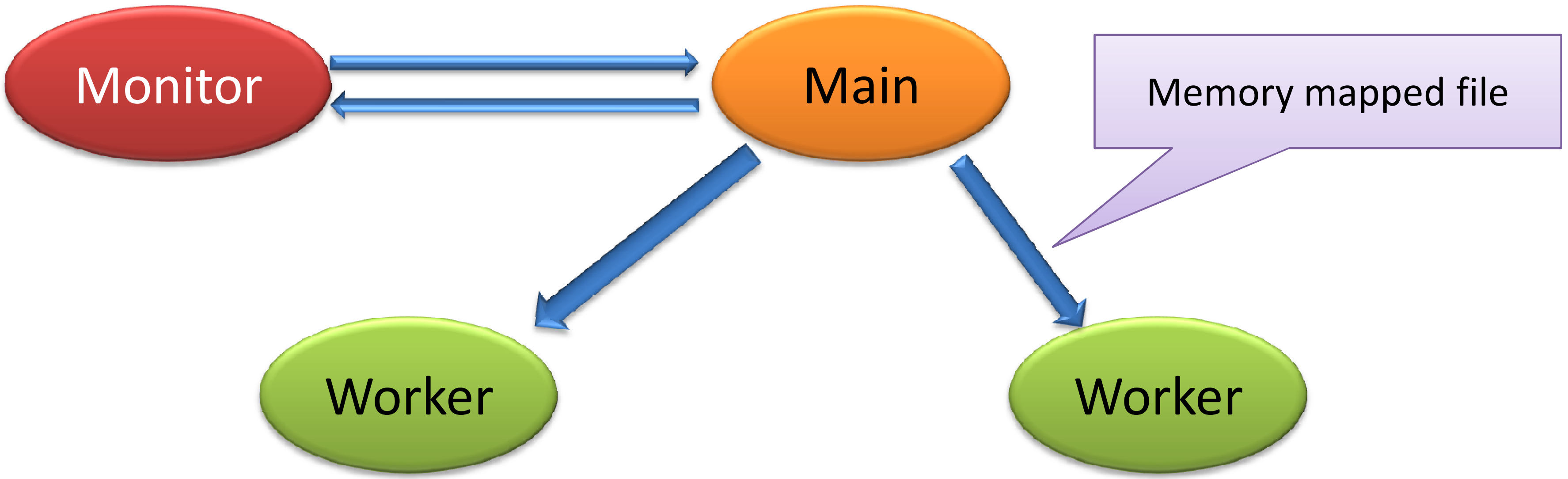
```
for(i = 2; i < argc; i++){
        sem_wait(&(shm_p->sem_slots);
        sem_wait(&(shm_p->sem_mutex);
        strcpy(shm_p->filename, argv[i]);
        printf("MONITOR %d has set a new file %s\n", getpid(), argv[i]);
        sem_post(&(shm_p->sem_mutex));
        sem_post(&(shm_p->sem_resources));
}

kill(main_process_pid, SIGUSR2);
```

Don’t write if there’s a filename already

There’s a new filename available

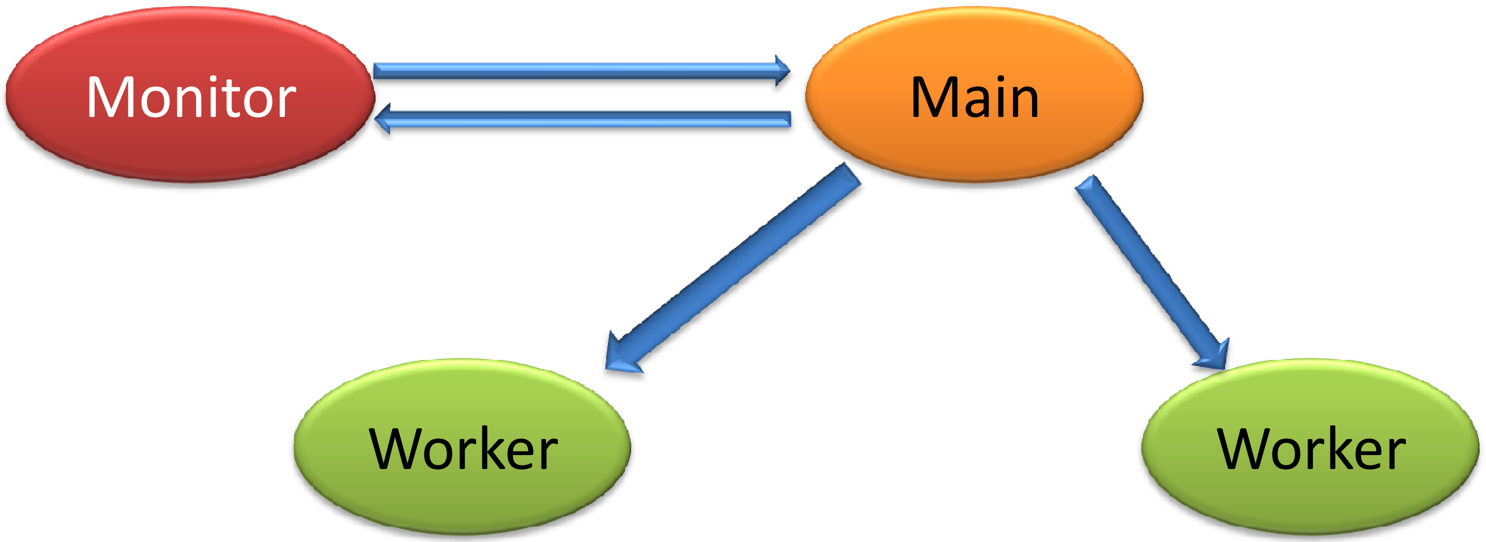
Done with all files

Processing a File



1) Prepare a file to store the results
2) Fork a worker for each file
3) Map in memory the file in both processes
4) Wait for WORKER to finish
5) Send results to Monitor through pipe

Processing a File

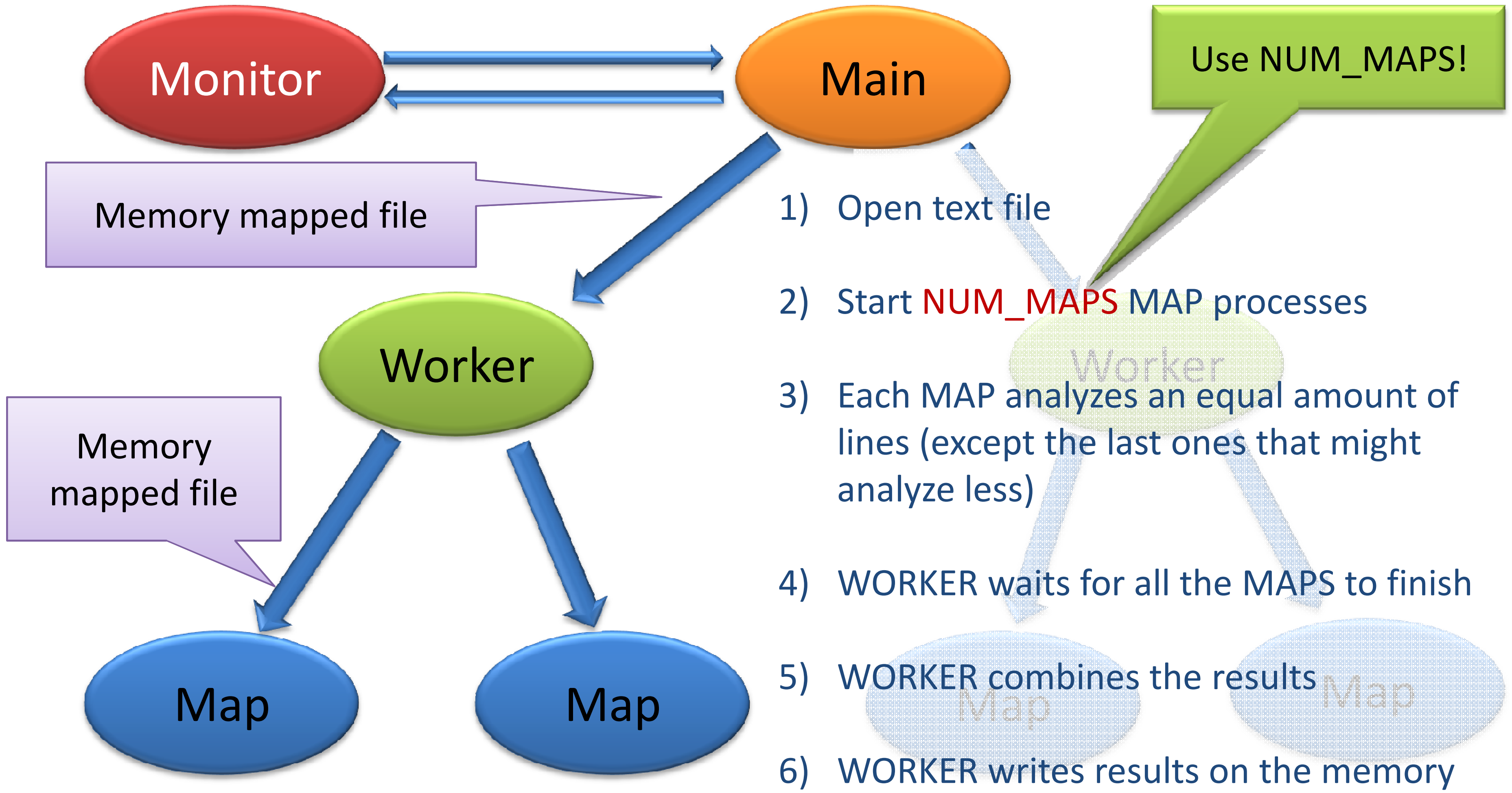


1) Prepare a file to store the results
2) Fork a worker for each file
3) Map in memory the file in both processes
4) Wait for WORKER to finish
5) Send results to Monitor through pipe

DO NOT BLOCK!!!

Remember that when you fork() you create a copy of all your memory until now

Map-Reduce



1) Open text file
2) Start NUM_MAPS MAP processes
3) Each MAP analyzes an equal amount of lines (except the last ones that might analyze less)
4) WORKER waits for all the MAPS to finish
5) WORKER combines the results
6) WORKER writes results on the memory mapped file shared with MAIN

Mbox Files

```
H  From - Tue Mar  9 19:29:41 2010
N  X-Mozilla-Status: 0001
N  X-Mozilla-Status2: 00000000
N  Path: dcs-news1.cs.illinois.edu!not-for-mail
N  From: "[TA] Wade Fagen" <cs241help-sp10@cs.illinois.edu>
N  Newsgroups: class.sp10.cs241
N  Subject: Re: Anyone get mp1 grades?
N  Date: Mon, 15 Feb 2010 02:05:41 -0600
N  Organization: Department of Computer Science, University of Illinois
N  Lines: 4
N  Sender: wfagen2@gng0159.urh.uiuc.edu
N  Message-ID: <hlav9v$4dk$1@dcs-news1.cs.illinois.edu>
N  References: <hl9ior$i3l$1@dcs-news1.cs.illinois.edu> <hl9j5s$ieo$1@dcs-news1.cs.illinois.edu>
N  <hl9j80$igf$1@dcs-news1.cs.illinois.edu> <hl9lbk$kr0$1@dcs-news1.cs.illinois.edu>
N  NNTP-Posting-Host: gng0159.urh.uiuc.edu
N  Mime-Version: 1.0
N  Content-Type: text/plain; charset=ISO-8859-1; format=flowed
N  Content-Transfer-Encoding: 7bit
N  X-Trace: dcs-news1.cs.illinois.edu 1266221183 4532 130.126.80.68 (15 Feb 2010 08:06:23 GMT)
N  X-Complaints-To: abuse@cs.illinois.edu
N  NNTP-Posting-Date: Mon, 15 Feb 2010 08:06:23 +0000 (UTC)
N  User-Agent: Thunderbird 2.0.0.23 (Windows/20090812)
N  In-Reply-To: <hl9lbk$kr0$1@dcs-news1.cs.illinois.edu>
B  Xref: dcs-news1.cs.illinois.edu class.sp10.cs241:760
M  The autograder results are now in your svn.  There'll be an announcement
M  in the announce newsgroup in just a minute.
M  - wade
M
```

Things You Might Want To Know - Signals

```
struct sigaction usr1_action;
usr1_action.sa_handler = usr1_handler;
sigemptyset (&usr1_action.sa_mask);
usr1_action.sa_flags = 0;
sigaction(SIGUSR1, &usr1_action, NULL);
```

- SIGCHLD, generally ignored, is signaled to the parent when a process exits
- When SIGCHLD is signaled, the process is a zombie, waiting for a waitpid() call from the parent.
- Signals are not reliable
  - If more then one (of same type) arrives at the same time, the handler might be called only once

Things You Might Want To Know - Signals

- When forking a new process you might need to change the way the new process handles signals. You can revert to the default handler with:

```
struct sigaction chld_action;
chld_action.sa_handler = SIG_DFL;
sigemptyset (&chld_action.sa_mask);
chld_action.sa_flags = 0;
sigaction(SIGCHLD, &chld_action, NULL);
```

Things You Might Want To Know – Shared Memory

- The system allows only for a limited number of segments
- When you are done with the shared memory you MUST remove the mapped segment

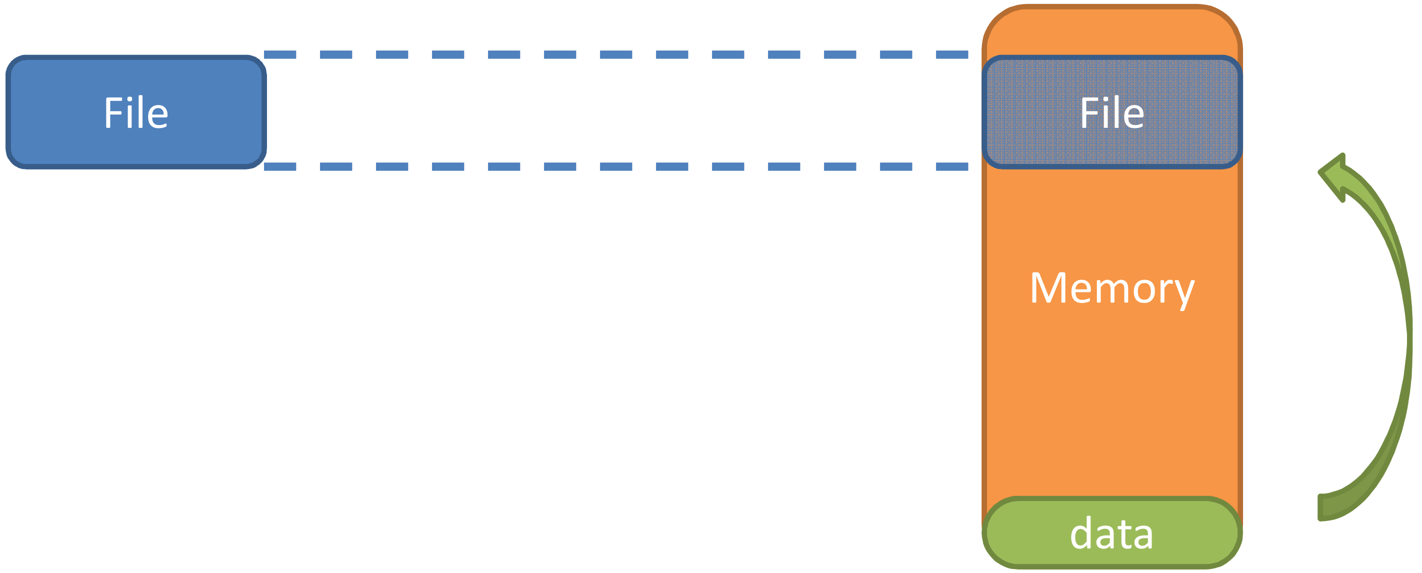
```
shmctl(shmid, IPC_RMID, (struct shmid_ds *) NULL
```

- This just MARKS the memory to be destroyed, it will happen only when the last process detaches it

Things You Might Want To Know – Memory mapped files

- Reading and writing to a file as if it was a memory location
- Don’t have to worry about write cache and delays

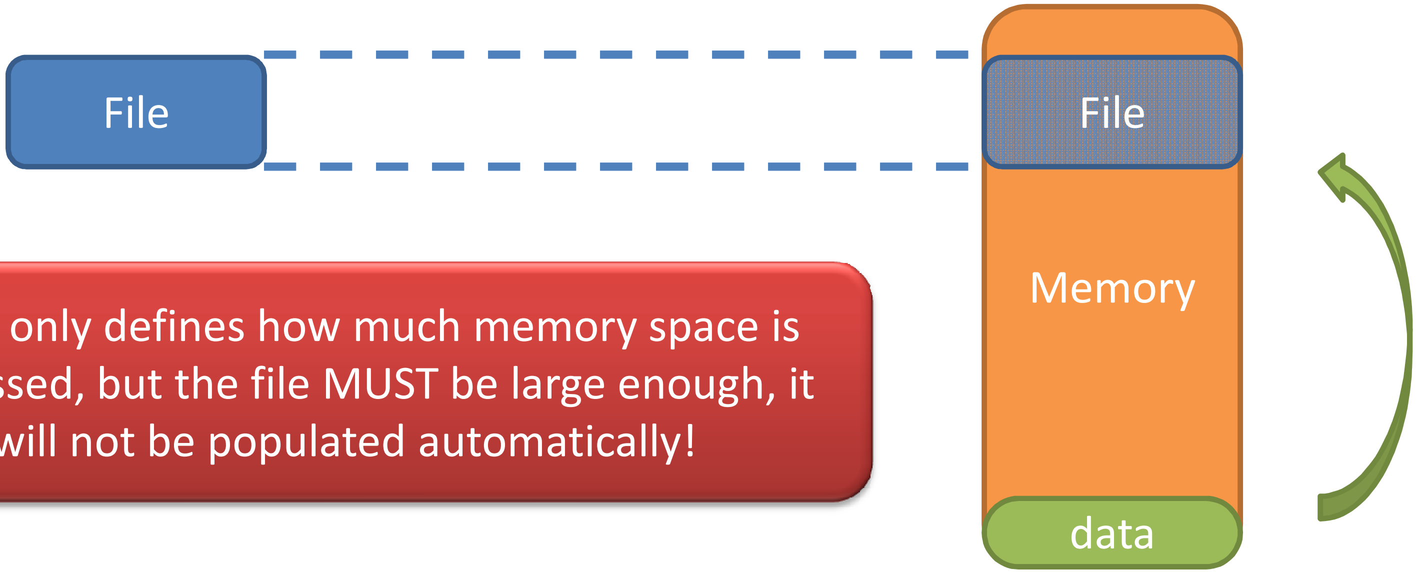
```
int mmappedfile = open(mmapfilename,O_RDWR|O_CREAT,0666);
char* data = mmap((caddr_t)0, size, PROT_READ|PROT_WRITE,
MAP_SHARED,mmappedfile ,0);
data[0] = ‘\0’;
```



Things You Might Want To Know – Memory mapped files

- Reading and writing to a file as if it was a memory location
- Don’t have to worry about write cache and delays

```
int mmappedfile = open(mmapfilename,O_RDWR|O_CREAT,0666);
char* data = mmap((caddr_t)0, size, PROT_READ|PROT_WRITE,
MAP_SHARED,mmappedfile ,0);
data[0] = ‘\0’;
```



“size” only defines how much memory space is addressed, but the file MUST be large enough, it will not be populated automatically!