Operating Systems - Assignment 2

Ansgar Lemke (ansgar.lemke@stud-mail.uni-wuerzburg.de, ens21ale)

Abstract

A simple scatter-gather functionality will be implemented using Linux
processes only.

1 Description

scatter-gather is an algorithm to split up a memory area so that multiple processes can work on the same computational problem and combine the results to one field of memory afterwards. MPI implements this functionality as MPI_Scatter_Gather, however, this solution aims to be as easy to use as possible. It's only dependencies are standard libraries.

1.1 Solution

The library is implemented in C99 which makes it available to many applications. With the used of shared memory but separated areas within for the different processes, questions of synchronization fall apart in most of the implementation. In the scattered phase, the processes work independent of each other and do not need any synchronization or communication which offers a good concurrent environment.

1.2 Design Decisions

Each process is assigned to a segment which corresponds to a certain area of the shared memory. The highest segment is assigned to the parent process. This information can be used to exit all other processes in the end to continue on the parent only.

One goal is to provide an interface to the user which grants full functionality while being relatively safe by hiding parts that are not relevant for end users. There are two functions available to be used by the user:

- scatter takes the data where the processes should work on (init_data), the number of processes (segments) to create, a pointer to the memory for each process (proc_data) and the length of the whole data in bytes. It returns an identifier for each process (no pid) or -1 in case of error.
  scatter creates an area of shared memory, copies init_data into this area, forks, sets the proc_data pointer of the child to a position inside this and returns. For the parent, the last segment of the memory will be used.

- gather takes only one argument, a pointer where to write the combined work of the processes (exit_data). If gather works fine, it returns 0, otherwise the number of processes that have not finished from what reason ever. For example, when gather is called a second time.
  gather exits the child processes while letting the parent wait for all children to exit and then copies the shared data back into exit_data followed by return.

To transmit data from scatter to gather, internally a static struct is used. It contains the following values

- com - the shared memory

- segment - the identifier of the process

- length - size of shared memory in bytes

- segments - number of parallel processes

- used - a state variable to prevent multiple scatter calls

The above mentioned pointer to the memory for each process points to a location inside the shared memory so some memcpy calls can be avoided. In the end of gather, the content of the shared memory is being copied back to exit_data so that no shared memory exists after gather has finished.

1.3 Assumptions

The spawned processes must only work on their own memory section, proc_memory. They must not change the location of the pointer because then their changes will not be written back. They must strictly remain inside their memory area, otherwise they would write either on memory of another process or even to an invalid location outside the shared memory area.

init_data must always be initialized while the other pointers should not, but may be as well (causing memory leaks).

It is assumed that all processes take nearly the same amount of time, otherwise, the waiting might take up to the time the slowest process needs to finish plus the time to exit the others.

1.4 Limitations

The amount of available data types is restricted to the basic C types which do not contain any pointers to other memory locations and have a fixed size, e. g. int, char. Inclusion of pointers would introduce the problem that these need to be copied to the shared memory as well while it is not known how their structure looks like.

The scatter-gather calls cannot be nested, if scatter is called, gather must follow exactly one time. This greatly simplifies the structure because a lot of the functionality can be hidden from the user.

Of course, there is a maximum size of memory that can be used in RAM.

On multi-processor systems, the behavior of cache pages might reduce performance when data needs to be copied back again or adjacent memory areas might be synchronized as well out of the interpretation that most users might want to acces these as well.

2 Examples

To demonstrate the functionality of scatter-gather, two computational problems will be executed with different amounts of processes.

2.1 a

bla.

References

[1] Corbet, Jonathan, Schedulers: The Plot Thickens, LWN.net, (April 2007, https://lwn.net/Articles/230574/).

[2] sched(7) Linux User's Manual, Edition 5.3, (February 2022).

[3] oxnz: Linux Asynchronous I/O (October 13 2016, https://oxnz.github.io/2016/10/13/linux-aio/).

[4] open(2) Linux User's Manual, Edition 5.13, (February 2022).

[5] colin-king et al: IOSchedulers ubuntu wiki (September 2019, https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers).