Bachelorarbeit

Hochschule
Augsburg University of
Applied Sciences

Fakultät für Informatik

Studienrichtung

Technische Informatik

Florian Pîrvu

Arbeitstitel: C++ Crossplatform Bibliothek zur Performanceanalyse von Anwedungen unter dynamisch generierten Lasten

Erstprüfer: Prof. Dr. Thomas Kirchmeier

Zweitprüfer: Prof. Dr. Hubert Högl

Abgabedatum: 20.01.2022

Hochschule für angewandte
Wissenschaften Augsburg

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info(at)hs-augsburg-de

Fakultät für Informatik
Telefon +49 821 55 86-3450
Fax   +49 821 55 86-3499

Verfasser der Bachelorarbeit
Florian Pîrvu
Asternweg 2
86399 Bobingen
Telefon +49 176 3693 7974
pflorian306@gmail.com

Inhaltsverzeichnis

1. Introduction

1.1 Testing

Software testing refers in the IT-Branch to a product with the main goal of finding bugs in a software program or application. These bugs refer to errors or faults and can occur because of a bad commands sequence specified in the program's source code.

A successful test can be achieved when its main requirements are met. Some examples would be the execution on different environments, time constraints or delivering expected outputs for randomly chosen inputs. Conditions are set by the tester and may vary depending on the use case.

1.2 Motivation

Most companies put a lot of effort in delivering high performance products to their customers. In order to do so, each of them test their gadgets, machines or software for possible failure scenarios. Now-a-days tests are being fully automated, which decreases the failure possibility that can happen because of human errors.[1]

Unfortunately the creation of fully automated tests is not as easy as it may sound. Many testing developers know the struggle of finding the right tools for the job and by the end of testing phase, their project is filled with unnecessary dependencies that will overload the program and occupy valuable memory.

"As ironic as it seems, the challenge of a tester is to test as little as possible. Test less, but test smarter"

Federico Teldo, Co-Founder Abstracta US

Additionally the problem enhances when a company reaches a certain size with a significant number of customers, which tend to run the product on different machines and architectures. In order to keep their customers, producers need to adapt their products to support newer or older machines. This makes software analysis even more difficult because of the increasing complexity, which comes with different systems. For this reason many companies dedicated themselves to creating programs that focus only on software examination and fixing. In time a new trend has been created with demands so high that rapidly developed itself into a new market section.

1.3   Growing markets

For the last decade the software testing market has been developing and now it has grown so big that it would be foolish to ignore it. According to "Global Market Insights" the testing software market has grown up to 40 billion dollars by the end of 2020 and is predicted to grow up to 60 billion dollars in the next 6 years.[2]

"People may lie, but number don't". Multiple scientific papers and studies enforce this statement with different statistics of industries, which started adapting and reacting to this trend using the model "EaaS " which stands for "Everything as a service" and created "Testing as a Service"(TaaS). With this model customers not only pay for the current state of a product, but also for a service subscription, where they get updates and new features for the specific product and additional support from the company. The advantages that benefit the customer are set by the company for each of their subscription. (The basic rule is that you get more accurate results if you pay more). This service usually targets three groups: Developers, End Users and Certification Services.[3]
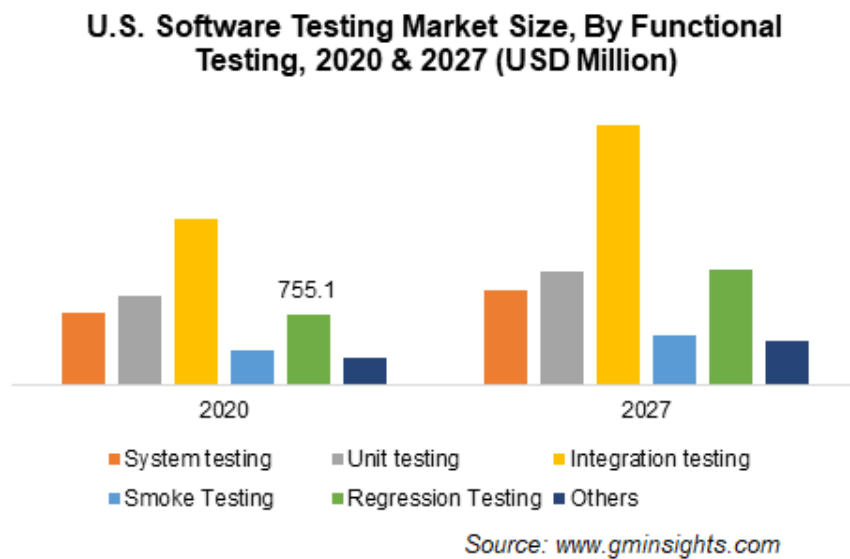

Abbildung 1.1: US Software Market Size[2]

1.3.1   Cost comparison

People often tend to overlook costs of testing tasks that are insignificant in comparison to the total price of the project. However if these costs are reoccuring, their total price can go up to 40% of the total project's costs[4]. That amount of money covers not only for direct testing, which includes the staff, system and program testing, resources, computer time, etc. , but also for indirect testing, which refers to actions that take place because of poor direct testing, like rewriting code, additional analysis meetings or debugging.

It was proven that the cost of finding an error is about $50 on average [5], and is said that fixing an error after the software was released is four times more expensive than compared to if it was found during the testing phase.[6] To avoid these financial expenses, companies train their developers to consider failure scenarios of the product early in the development stage.

1.3.2   Common approaches

In order to deliver a defect-free product, managers create a testing strategy. To develop the most suitable strategy, they must identify the key components for it . This can be identified mostly by answering the following questions adapted from [6]:

1. Is the objective clear specified?

2. What tools will be used?

3. Is the system fully/partially automated?

4. How will the test benefit the project?

These questions should always be asked when a new feature that needs testing is being developed.

1.4   Current Situation

At the time of writing this (December 2021), there are four mostly used architectures in the IT-Branch. These are Windows, Linux and MacOS. Linux was developed with the UNIX system as its core, while MacOS is only based on UNIX, which means that these are similar but not entirely compatible with each other. The big difference comes with Windows.
Each operating system can deliver informations about its own computer.

For that each of them has its own unique program. Windows uses the "Task-Manger" which comes with a GUI and shows real-time information about the CPU and memory of the computer, it comes with a list of processes and makes it possible for the user to manage them with only a couple of clicks.
Linux on the other hand is more text oriented. This operating systems comes with a built in command called "top". This also delivers informations about your system, but it doesn't allow you to manipulate processes like "Task-manger" does. Although not very practicable for developers, these tools allow the user to measure performance in a normal state where your computer is not put under pressure, but the most interesting state is when the CPU needs to do a lot of operations and it needs to share its valuable time with other processes. To force such situations, the user can use online stress tests, which use the browser as an workload source, which for many is not very practicable, because it uses an additional program, which runs in the background. Online stress tests rely on internet to work and a stable ethernet or wifi connection to work. This way the results can be very inaccurate if the the network is under pressure.
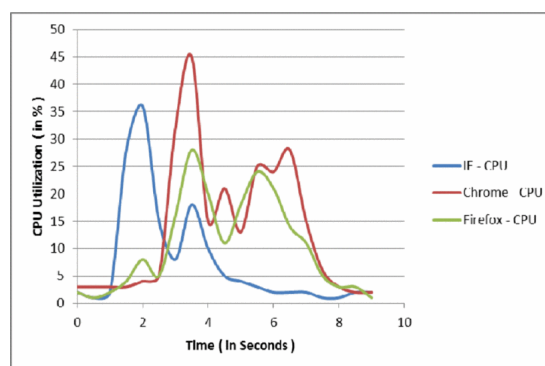
Abbildung 1.2: Browser CPU utilization comparison[7]

There are already some alternative libraries on the internet to recreate this method, but most of them are written in other programming languages. Adding them to a C++ Project would rise the complexity of the source code.

---

[0]Some companies also use ARM(aarch64), but this is not an OS and per default doesn't come with any process control system. In order to do so one needs to write or use an extra library for that solely purpose

Some people would think "Well C++ is a new and modern language. There must be something similar out there". This statement is true and because of its young age there are a limited number of officially tested methods. You could also take a look at the headers implemented in the C++ standard library [8] to familiarize yourself with common algorithms and data structures.

1.5   Obective

The purpose of this thesis is the development, implementation and testing of a Library for C++ applications that creates an artificial workload and delivers system independent statistics based on multi-threading. This will allow developers to implement their own stress tests and run simulations with different workloads, process priorities and scheduling, which can be used in different manners to achieve real-time applications and create benchmarks for each new product. These could be later automatized by using programs like Jenkins[9] or Gitlab[10].

1.6   Implementation

The library has two main components. The first one is the workload, which can vary depending of the user's input. This will create a system specific amount of threads and make them run a simulation function. The number of threads will stay constant and the time for running the workload task will be set accordingly for each input.
The second components is the system. This comes with functions, which can easily change a thread's priority, a system's scheduling policy and deliver statistics of the user's computer. Basic operations are implemented to work on most operating systems, but there are some exceptions because of the differences between OS implementations, which makes them independent from one another.

1.7   Advantages and Disadvantages

1.7.1   Pros

1. No additional expenses for third party software or subscriptions

2. Can be directly added to the source code, which makes testing easy

3. No external influence from the browser(ads) or internet connection, allowing efficient testing even offline

4. Allows the creation of tests based on user experience

5. Companies don't have to give their data

1.7.2   Cons

1. Tests don't come already prepared

2. Developers have to add the source code to their build system or add the build files(CMakeLists.txt) to their own (if they already use CMake)

1.8   Summary

With this library a company can save money, improve their product based on user feedback and create an internet independent test system for very minimal effort.

2. System fundamentals

Not all operating systems will be covered in this thesis. The following statements are true for the testing machines used in chapter 4?.

2.1 Processes

To many operating systems a process is like an wrapper defined by the kernel in order to allocate resources to an executing program.

When a process is created the system assigns him a *process unique identifier* also called PID(a positive integer). Each process has its own PID, so two different executing programs cannot have the same PID [1]. The methods used to create a new process differ on each operating systems. On Windows you can create processes by calling the `CreateProcessA()` function [11], which returns a HANDLE (the equivalent PID for windows systems) and on linux you can use `fork()`[12]. Each operating system calls one these methods internally every time the computer or the user starts a routine of execution, like starting a service, or a program (browser, spotify, etc). You can think of this system like a binary tree with one or more children with the kernel on top. The children have also an additional attribute called PPID, that contains the PID of its creator. If this is specified to zero then the kernel is the parent.[13][2].

2.1.1 Threads

Each process has at least one thread of execution called the main thread, which as the name say contains the `main()` function. Once a thread has been created, the main thread has to wait for it to finish by calling a method called `join()`. As their creators, threads also have unique identifiers called *Thread identifiers* or TID and OS-specific methods to create them. On UNIX systems one would use the `pthread_create()` and on windows `Create_Thread()`. But the role of this library is to make this whole process as easy as possible, so we will use the C++ Standard Library's threads (std::thread) and "detach"[3] ourselves from the other ones.

Processes have their own stack, so they can't communicate with each other. This is a huge problem when it comes concurrency (explained in **??**), but threads share the stack of their creator-process.

---

[1] On UNIX like machines the first process to be called is the init process with PID 1

[2] There a very few processes that have the PPID=0 (ex. init)

[3] There is also a method called `detach()` which allows a thread to separates itself from the main thread. This way the thread can terminate itself and the main thread doesn't have to wait for that thread's termination

2.1.2   Differences between threads and processes

Many people tend to think of threads and processes as being the same, but they are quite different. First as mentioned above threads share the same stack of the creator-process, while processes need intercommunication tools like pipes to talk to each other. Another difference is that a process can have multiple threads attached to it, but a thread cannot belong to more than one process.[4]. Their identifiers are also independent from their creator's. This way a TID can be equal to its creator's (or another process's) PID.
One can imagine a process like an octopus and the threads being it's arms. One octopus has many arms that can execute multiple tasks at the same time, but an arm cannot belong more than one octopus.
In this library we use multiple threads to simulate a user specific workload because this way we can time their execution start point with only one shared variable and we don't have to worry about interprocess communications.

2.1.3   Attributes

Normally when we would create a threads using the unix methods, we would pass a pointer to a structure that describes the attributes for that specific thread(that structure can be created with `pthread_attr_init()` and be destroyed with `pthread_attr_destroy()`. Some of these attributes include the scheduling priority, scheduling policy and stack size, which are important for out tests. Unfortunately the standard library doesn't have this option. In order to set and get this attributes we need to use the architecture's dependent functions.

2.1.4   Stack Size

The stack is a piece of memory where meta-data and local variables are saved when the `main()` function calls a routine/method. This memory segment is limited and doesn't allow an infinite number of data segments (also called stack frames) being stored in it. The stack uses assembly instruction like `pop` to delete a frame and `push` to add a frame. For consistency the stack will always pop the last element pushed. This is also known as "Last in First Out"[14].
On UNIX systems we would create an attribute structure and pass the desired options

Abbildung 2.1: Stack segment
[14]

there. But because we don't use the unix's system function `pthread_create()` we also cannot use the attribute structure. Furthermore the standard library doesn't support such tweaks. This is very important if someone wants to use this for an ARM architecture, because he won't be able to define a meaningful stack size. This doesn't pose any threats for many operating systems out there, but for ARM, which has a limited stack size can be problematic. For windows this attribute can be set using the `Create_Thread()` function just like in unix using `pthread_create()`.
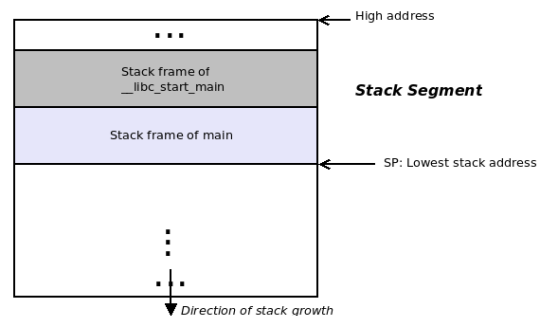
---

[4]If the execution method of a thread calls `fork()` then that process's PPID will be the PID of thread's creator

2.2 Concurrency

Concurrency means that two or more things are happening simultaneously. This pehnomenon is happening everyday almost everywhere we look. Even we as humans are capable of such thing, for example walking and talking at the same time. In computer science concurrency means that more than one process can be execuvted at the same time. Many systems have this ability, because most of them are multiprocessor computers. Even some single core computers can handle concurrency to some extent. One calculation unit can handle one task at a time, but it can quickly switch to another task if necessary. This is why sometime even single core units give the impression of resolving jobs simultaneously[15]. Although performant this method does not come without its flaws. When a cpu gets a new task, the resources of the old task (eg. local variable, meta data of the current task, etc.)will be replaced by those of the new job. This is also known as "Context Switching". The order of task switching will be explained in detail in chapter *scheduling*.
Nowadays many systems measure the concurrency of a system by its number of hardware threads. This unit of measure tells us how many independet tasks can the processor handle.

2.3 CPU Affinity

The affinity of a CPU determines the number of processors that one process can use for its threads. This library offers methods to restrict the number of CPUs off which a process can run on. This is sometimes desirable because of the following reasons:

1. Data invalidation: When a process starts, the user cannot tell from outside on which CPU that thread started. When a process finishes its time-slice, he has to give up its CPU for others to use it and it come back later, but it won't necessarily start on the same CPU as the last time. [5]. When this happens, entries in that CPU's cached must be replace or removed, which is known as "Cache invalidation". This is not a flawless method and cache inconsistencies can appear.

2. Emergency CPU: On real-time systems, where human lives are at risks, many developers will deliberately block some CPUs (on a multicore machine) to use them when the system returns erros and needs to immediately execute safety protocols. Because the CPUs were blocked from being used on other processes, these remain free and so the execution of the safety procedure can start without any delay (eg. context switching).

By default most systems allow each process to use all CPUs. If the user turns off half of the CPUs of a given process and tries to create an additional workload with the library's methods for that process, he must keep in mind that the workload will also be cut in half because the threads have less processors to work on.

2.4 Priorities

When it comes to the priority of a process there is a big difference between a UNIX system and a windows machine. On windows the priority is determined by the priority class of the process and the its thread priority.

---

[5]This is a part of context switching, which was discussed in chapter 2.2

2.4.1   Windows

Based on the winAPI documentation[16], the classes can have the following values:

1. IDLE_PRIORITY_CLASS (0x00000040): This is the lowest priority, processes belonging to this class run only if the system is idle and can be preempted[6] by a process with a higher priority

2. BELOW_NORMAL_PRIORITY_CLASS(0x00004000): This class has a higher priority than an idle-classed process but a lower priority than a normal-classed process

3. NORMAL_PRIORITY_CLASS(0x00000080): This is the default class for all processes created by the user

4. ABOVE_NORMAL_PRIORITY_CLASS(0x00008000): This class has a higher priority than an normal-classed process but a lower priority than a high-classed process

5. HIGH_PRIORITY_CLASS(0x00000080): This class is usually used for time critical jobs

6. REALTIME_PRIORITY_CLASS(0x00000100): This is the class with the highest priority and is rarely used because it stop most of the tasks on the calling machine

Each class can be preempted by a higher priority class besides the realtime-class. Classes categorize only processes, but not their created threads. For these the following values can be set:

1. THREAD_PRIORITY_IDLE(-15)

2. THREAD_PRIORITY_LOWEST(-2)

3. THREAD_PRIORITY_BELOW_NORMAL(-1)

4. THREAD_PRIORITY_NORMAL(0)

5. THREAD_PRIORITY_ABOVE_NORMAL(1)

6. THREAD_PRIORITY_HIGHEST(2)

7. THREAD_PRIORITY_TIME_CRITICAL(15)

These are similar to the classes mentioned above and can be interpreted alike.

2.4.2   Linux

On Linux however, the priority of a process is harder to be determined. This value is composed out of two main components: the nice value of the process and its thread priority.

---

[6]If a process is preempted that means it stops executing and yields the cpu

2.4.2.1   Nice Values

Nice values can range from 20 to -19 with 20 being the nicest value and so the smallest priority and -19 being the worst value and so the highest priority. For a better understanding one could think that one process is nice when he doesn't need the CPU and so it lets other threads to use it. In my reasearch one thing was mentioned and that is that a low nice value (hence a high priority) doesn't mean that other processes won't get any CPU time. The scheduler will make them favorable, but other processes will also get their turn for the CPU. To change the nice values of a process in this library i am using the calls `getpriority()` and `setpriority()` from the header `sys/resource.h`.

There is one critical thing that the caller needs to know. In order to increase the nice value of the calling process, the user needs to use the given methods of the library and additionally use the command `sudo setcap cap_sys_nice=ep PATH/TO/EXECUTABLE` on the built binary, run it as root or build the executable program as the root-user. You need to do this extra step because by default any user-created processes are unprivileged.

Unprivileged processes can lower their own priority, but are not allowed to increase it more than the value of the operation *20-RLIMIT_NICE*. The RLIMIT_NICE is resource on your UNIX machine. The structure `rlimit` has two attributes: the "rlim_cur"(also called the soft limit), which represent the current value of the process and the "rlim_max"(also called the hard limit or ceiling), which tells one user the limit of which that process can be set to. On my testing system RLIMIT_NICE is set to 13 and the limits for processes compiled by my users are zero.

```
101 int main(int argc, char* argv[])
102 {
103         struct rlimit oldcap, newcap;
104         std::cout << "getrlimit: " << getrlimit(RLIMIT_NICE, &oldcap) << std::endl;
105         std::cout << "rlimit_nice soft: " << oldcap.rlim_cur << std::endl << "rlimit_nice
    hard: " << oldcap.rlim_max << std::endl << "RLMIT_NICE: " << RLIMIT_NICE << std::endl;
106         return 0;
107 }
```

(a) RLIMIT_NICE code

```
element@element-Inspiron-15-3567:~/Desktop/Florin/Bachelorarbeit/performance/but
ld(main)$ ./bin/test
getrlimit: 0
rlimit_nice soft: 0
rlimit_nice hard: 0
RLMIT_NICE: 13
```

(b) RLIMIT_NICE output

Abbildung 2.2: RLIMIT_NICE

Per default the process will have the nice value of 0 and this value can be increased to the highest value allowed (19), but cannot be decreased afterwards to a value lower than *user_nice_value = 20-RLIMIT_NICE*. The command `setcap` can change the executable to run as a privileged process (but only when called as root or with the keyword sudo). A way of increasing the nice value would be to increase the RLIMIT_NICE value (also as root or with root-rights = sudo), which will allow a normal user to increase the value given until it reaches "user_nice_value" or log in as root, use the library's functions, build the program and set the SUID as root. [7]

At last you could also modify the "/etc/security/limits.conf" file and set a new max nice value for a certain user, but this is not the best solution, because that user would have the power to change priorities not only for one program, but for all programs.

---

[7]The SUID is a special bit that one can set and allows normal users to run the program as they were root

2.4.2.2   Scheduling

Unlike Windows, Linux has methods to change one's process and its threads scheduling policies.The default policy set on UNIX is called "Round-Robin Timesharing"(SCHED_OTHER). This allows jobs to be executed in a round robin fashion where each process gets an equal time-slice of a CPU. There are more than one policy which can be set. These are:

1. SCHED_OTHER

2. SCHED_BATCH

3. SCHED_IDLE

4. SCHED_FIFO

5. SCHED_RR

The difference between SCHED_RR and "Round Robin Timeshare"(SCHED_OTHER) is that the realtime policy lets us to coordinate the priorities for that schedueling policy's queue.

The differences are that BATCH schedules a process less frequently, if the it gets the CPU very often and IDLE is the equivalent to a process with a nice value of 19 (very nice process $<=>$ lowest value). SCHED_BATCH and SCHED_IDLE are two normal prioritised policies, which differ from SCHED_OTHER, but not enough for me to focus too much on them.

You can get the current policy of your process by calling `int sched_getscheduler(pid_t pid)` from the `<sched.h>` header file.

Each of the policies mentioned above has a range of priorities levels, which can be get using the `sched_get_priority_max(int policy)` and `sched_get_priority_min(int policy)` methods found in `<sched.h>`.[8]

On my Linux Notebook I have the following values: You can only change the priority of


```
std::cout << "sched_get_prio_min fifo: " << sched_get_priority_min(SCHED_FIFO) << std::endl;
std::cout << "sched_get_prio_max fifo: " << sched_get_priority_max(SCHED_FIFO) << std::endl;
std::cout << "sched_get_prio_min rr: " << sched_get_priority_min(SCHED_RR) << std::endl;
std::cout << "sched_get_prio_max rr: " << sched_get_priority_max(SCHED_RR) << std::endl;
std::cout << "sched_get_prio_min other: " << sched_get_priority_min(SCHED_OTHER) << std::endl;
std::cout << "sched_get_prio_max other: " << sched_get_priority_max(SCHED_OTHER) << std::endl;
std::cout << "sched_get_prio_min idle: " << sched_get_priority_min(SCHED_IDLE) << std::endl;
std::cout << "sched_get_prio_max idle: " << sched_get_priority_max(SCHED_IDLE) << std::endl;
std::cout << "sched_get_prio_min batch: " << sched_get_priority_min(SCHED_BATCH) << std::endl;
std::cout << "sched_get_prio_max batch: " << sched_get_priority_max(SCHED_BATCH) << std::endl;
```

(a) sched_prio_range code


```
[100%] Built target test
element@element-Inspiron-15-3567:~/Desktop/Florin/Bachelorarbeit/performance/build(main)$ ./bin/test
sched_get_prio_min fifo: 1
sched_get_prio_max fifo: 99
sched_get_prio_min rr: 1
sched_get_prio_max rr: 99
sched_get_prio_min other: 0
sched_get_prio_max other: 0
sched_get_prio_min idle: 0
sched_get_prio_max idle: 0
sched_get_prio_min batch: 0
sched_get_prio_max batch: 0
```

(b) sched_prio_range output

Abbildung 2.3: sched_prio_range

real-time policies. hence when you set the policy of a thread to OTHER, IDLE or BATCH you can't change the priority of that process.

As you can observer only two of these have a "real" range. SCHED_RR and SCHED_FIFO

---

[8]These can be different depending on the calling xUNIX machine

are characterized as real-time policies and have a higher priority than the others. When two processes with SCHED_RR and respectively SCHED_FIFO have to share a CPU, ironically the one that was placed first in that CPU's queue will get to run its job. Both of these policies can lose access of their CPU, if they finish execution, yield_sced() or a syscall is called and a higher priority process preempts them (a process with a lower nice value appears in the queue or the user changes the value himself). SCHED_RR can also lose its access if the timeslice of the job expires.

2.5 Workload

The definition for workload varies from field to field. In the IT-branch it is defined as a unit of measure for your CPU (mostly in %). This tells the user how well his system can handle the number of current running processes. Most systems calculate their workload over a defined period of time. To understand this concept better, here is an example. Let's say a user starts an calculator program at time $t_0 = 0$. The application will finish initializing a GUI at time $t_{wait} = 2$s and then wait for the user to enter some equation. After the equation was typed at $t_{input} = 5$s and the Ënterkey was pressed, the app proceeds calculating and delivering the answer at $t_{done} = 6$s. So the CPU-time of the application is 3s(GUI initialization and calculation time). In order to get its workload the system has to divide this time by the time of the system needed for the whole process $\frac{t_{wait}+t_{calc}}{6s} * 100$. A detailed explanation on how to get these values will follow in Chapter (*Kapitelnr*).

3. Beispiele

Bla fasel. . .

Beispiele

3.1 Zitieren

Quellen[?, ?, ?, ?, ?] nicht vergessen. Dazu verwendet ihr bibtex.
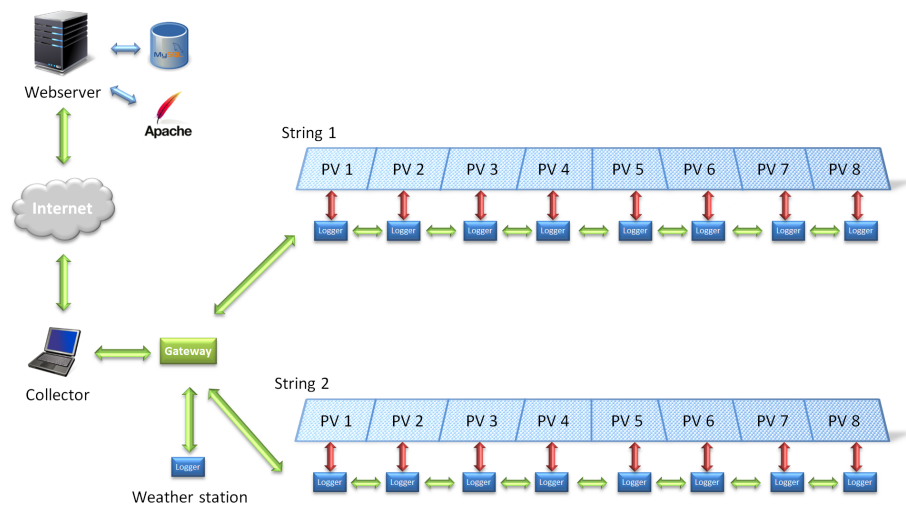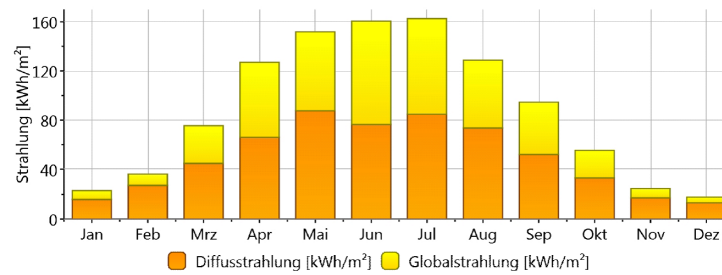
3.2 Bild einfügen

3.2.1 Ein Bild skaliert

Abbildung 3.1: Beschriftungstext

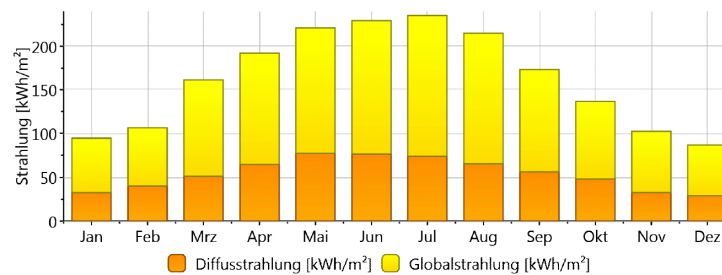### 3.2.2 Zwei Bilder nebeneinander oder untereinander

3.3 Tabellen

(a) Beschriftung Bild links

(b) Beschriftung Bild rechts

Abbildung 3.2: Beschriftung beide Bilder

Firma	Produkte / Lösungen	WEB
Concentrix (Soitec)	Module mit Konzentratoren (Fresnel-Linsen)	http://www.soitec.com
Isofoton	Module mit Konzentratoren (Fresnel-Linsen)	http://www.isofoton.com
Semprius	Module mit Konzentratoren (Fresnel-Linsen)	http://www.semprius.com
Azur Space	Mehrfach Junction Zellenhersteller	http://www.azurspace.com
Cyrium Technologies	Mehrfach Junction Zellenhersteller	http://www.cyriumtechnologies.com
Emcore	Mehrfach Junction Zellenhersteller	http://www.emcore.com

Tabelle 3.1: Hersteller von CPV-Produkten

Tabelle 3.2: Single-hop Scenario - Traffic Pattern

Pattern	Parameter	Distribution	Range/Values
Burst	Burst IAT	uniform	[9.9; 10.1] s
	Packets per Burst	constant	100
	Packet IAT	constant	0.02 s
	Packet Size	constant	1024 bit
	# Sources	-	2
	Offset	uniform	[0; 1] s
Single	Packet IAT	uniform	[0.9; 1.1] s
	Packet Size	constant	1024 bit
	# Sources	-	[10;20;30;40;50; 60;70;80;90;100]
	Offset	uniform	[0; 1] s

4. Analyse

Bla fasel. . .

4.1 Abschnitt 1

Bla fasel. . .

4.2 Abschnitt 2

Bla fasel. . .

4.2.1 Unterabschnitt

Bla fasel. . .

4.2.1.1 Unter-Unterabschnitt

Literaturverzeichnis

[1] Karuturi Sneha and Gowda M Malle. Research on software testing techniques and software automation testing tools. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pages 77–81, 2017.

[2] Shubhangi Yadav Preeti Wadhwani. Software Market Insights software testing market. https://www.gminsights.com/industry-analysis/software-testing-market, September 2021. Accessed: 2021-11-06.

[3] George Candea, Stefan Bucur, and Cristian Zamfir. Automated software testing as a service. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 155–160, New York, NY, USA, 2010. Association for Computing Machinery.

[4] Kevin J. Valle-Gómez, Pedro Delgado-Pérez, Inmaculada Medina-Bulo, and José Magallanes-Fernández. Software testing: Cost reduction in industry 4.0. In *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, pages 69–70, 2019.

[5] Edward F. Miller. Some statistics from the software testing service. *SIGSOFT Softw. Eng. Notes*, 4(1):8–11, January 1979.

[6] Vasundhara Bhatia, Abhishek Singhal, Abhay Bansal, and Neha Prabhakar. A review of software testing approaches in object-oriented and aspect-oriented systems. In M. N. Hoda, Naresh Chauhan, S. M. K. Quadri, and Praveen Ranjan Srivastava, editors, *Software Engineering*, pages 487–496, Singapore, 2019. Springer Singapore.

[7] Vishal Anand and Deepanker Saxena. Comparative study of modern web browsers based on their performance and evolution. In *2013 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5, 2013.

[8] Cpp Reference list of c++ standard library headers. https://en.cppreference.com/w/cpp/header. Accessed: 2021-11-06.

[9] Jenkins. https://www.jenkins.io/. Accessed: 2021-11-09.

[10] Gitlab. https://about.gitlab.com/. Accessed: 2021-11-09.

[11] Createprocess function. https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa. Accessed: 2021-11-12.

[12] Michael Kerrisk. *The Linux Programming Interface*. No Starch Press, 2010.

[13] Process identifier. https://en.wikipedia.org/wiki/Process_identifier. Accessed: 2021-11-12.

[14] Kaiwan N Billimoria. *Hands-On System Programming with Linux*. Packt Publishing, 2018.

[15] Anthony Williams. *C++ Concurrency in Action, Second Edition.* Manning Publications, 2019.

[16] Getpriorityclass function. https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-getpriorityclass. Accessed: 2021-11-15.

Abbildungsverzeichnis