Software Spezifikation

libipc++

II

Inhaltsverzeichnis

Kapitel 1

Einleitung

libipc++ wird als Programmierprojekt im fünften Semester des Bachelor Informatik an der FH Technikum Wien erstellt.

1.1 Zweck

Dieses Dokument soll die technische, funktionale und nicht-funktionale Aspekte des Projekts erläutern. Ebenfalls werden Performance-Kriterien, sowie Fehlerkriterien erläutert. Am Ende soll die *libipc++* den hier definierten Regeln und Anforderungen entsprechen.

1.2 Umfang

Für das Projekt werden als erste Schätzung 400 Stunden benötigt. Was pro Person dann 200 Stunden ausmacht. Wobei ungefähr die Hälfte dieser Zeit für Software-Tests zur Verfügung stehen muss. Ausnahme ist natürlich eine reibungslose Programmierung.

1.2.1 Versionierung

Als Versionierung wird ein Zahlensystem verwendet. Es gibt drei Zahlen in der Versionsnummer, die major number, minor number und patch level. Die Version ist dann wie folgt aufgebaut: <major>.<minor>.<patch>.

Falls es zu einer Revision eines Patches kommt, dann wird für interne Zwecke eine Revisionsnummer an der Version angehängt, diese sieht dann wie folgt aus: <major>.<minor>.<patch>.<revision>.

1.2.2 Statusberichte

Am Ende einer jeden Entwicklungswoche, wird ein Statusbericht angefertigt. Dieser Bericht beinhaltet dann die abgeschlossenen Themen der Woche eines jeden einzelnen Entwicklers beziehungsweise die Probleme die der Entwickler hatte.

Weiters wird eine Statistik mit den Bereits abgeschlossenen Arbeitspaketen und den noch übrigen angezeigt und eine Zeittabelle mit den bereits aufgebrauchten Stunden eines jeden einzelnen Entwicklers angezeigt.

1.2.3 Zeiterfassung

Ein jeder Arbeitsschritt der für das Projekt investiert wurde, wird in der Zeiterfassung protokolliert. Dies dient für spätere Zeitschätzungen, sodass man auf diese Ressourcen zurückgreifen kann.

1.2.4 Kanban

Als Management wird eine abgewandelte Form von Kanban verwendet. Wobei aufgrund der immanenten Studienzeit lokale Änderungen vorgenommen werden können. Diese werden dann im Statusbericht am Ende der Woche protokolliert. Im folgenden werden die Meetings vorgestellt:

Statusmeeting

Findet im unterschied zu Kanban nur einmal die Woche statt. Am Ende einer jeden Woche, in diesem Fall Sonntag, wird ein Treffen vereinbart in dem die Fortschritte der Woche oder etwaige Probleme besprochen werden.

Root Cause Analysis

Dieses Meeting findet gleich nach dem Statusmeeting statt. Hier werden Probleme genauer analysiert, vor allem dauerhafte Problemen oder Tickets die nicht zu lange in einer Station verweilen.

Operations Review

Diese Meeting findet einmal im Monat statt, idealerweise am Ende des Monats nach den Root Cause Analysis. Hier werden die angewendeten Methoden genauer analysiert und gegebenenfalls verbessert. Dieses Meeting macht nur dann Sinn wenn genug Daten für eine Verwertung gesammelt wurden. Falls nicht genug Daten vorhanden sind, dann wird das Meeting um ein Monat verschoben.

Board

Als Board dient waffle.io.

Tickettypen

Alle Tickets die in Kanban verwendet werden, wurden übernommen. Diese sind:
Expedite - Haben hohe Priorität und müssen sofort gemacht werden.
Fixed Date - Haben einen fixen Termin für die Fertigstellung.
Vage - Sind nachrangige Tickets mit geringer Priorität.
Standard - Hat normale Priorität und wird als first in first out (FIFO) (First In First Out) behandelt.

Expedite

Diese Tickets dürfen mit Vorrang behandelt werden. Sie zählen nicht zu der Limitierung der einzelnen Spalten und müssen sofort abgearbeitet werden.

Fixed Date

Dieses Ticket hat eine normale Priorität bis zur angegebenen Deadline. Falls das Ticket bis zu diesem Zeitpunkt noch nicht fertig ist, dann wird es zu einem *Expedite* konvertiert.

1.3 Erläuterungen und Begriffe

OS Betriebssystem

SRS Software Requirement Specification

MPC Multi processing

IPC Interprocess communication

API Application programming interface

FIFO first in first out

OOP Object oriented programming

IDE Integragted development enviroment

LVA Lehrveranstaltung

MSDN Microsoft Developer Network

MSMQ Microsoft Message Queues

RAM Random Access Memory

1.4 Verweise auf sonstige Ressourcen oder Quellen

FH Technikum Wien	`http://www.technikum-wien.at/`
Software Requirement Specification (SRS)	`https://de.wikipedia.org/wiki/Software_Requirements_Specification`
Integragted development enviroment (IDE)	`https://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung`
Kanban	`http://de.wikipedia.org/wiki/Kanban_(Softwareentwicklung)`
Project Repository	`https://github.com/chronos38/libipc-`

Statusbericht	Google Docs
Root Cause Analysis	Google Docs
Zeiterfassung	Google Docs
Kanban Board	GitHub - Waffle Kanban Management
C++ Referenz	http://en.cppreference.com/w/
Design Pattern	https://de.wikipedia.org/wiki/Entwurfsmuster
CMake	http://www.cmake.org/
Boost.Interprocess	http://www.boost.org/doc/libs/1_56_0/doc/html/interprocess.html
OpenMP	http://openmp.org/wp/
Beej IPC	http://beej.us/guide/bgipc/
MSDN	http://msdn.microsoft.com/en-us/default.aspx
Win32 Processing	http://msdn.microsoft.com/en-us/library/windows/desktop/ms684841%28v=vs.85%29.aspx
Win32 Pipes	http://msdn.microsoft.com/en-us/library/windows/desktop/aa365784%28v=vs.85%29.aspx
MSMQ	http://msdn.microsoft.com/en-us/library/ms711472%28v=vs.85%29.aspx
Win32 File Mapping und Shared Memory	http://msdn.microsoft.com/en-us/library/windows/desktop/aa366556%28v=vs.85%29.aspx
Google Test	https://code.google.com/p/googletest/

Kapitel 2

Allgemeine Beschreibung

libipc++ dient zum Einstatz für Interprozesskommunikation. Der Grund weshalb diese Bibltiothek existieren soll, ist dass es noch keine schöne Interprocess communication (IPC) Lösung gibt. Das Ziel dieser Bibliothek ist deshalb eine syntaktische schöne und konsistente Application programming interface (API) für Multi processing (MPC) und IPC zu bieten.

Die Bibliothek wird auf allen gängigen und modernen Betriebssystem (OS) lauffähig sein. Abhängigkeiten gibt es nur auf OS-Ebene, wodurch keine weiteren Programmbibliotheken notwendig sind.

2.1 Produktperspektive

Als Vergleich soll hier die Boost.Interprocess dienen. Die Boost.Interprocess ist zwar mächtig, jedoch syntaktische nicht sehr elegant und erfordert eine höheres Verständnis für den Endanwender. Im Gegensatz dazu soll *libipc++* eine ähnlich mächtige API anbieten, jedoch eine syntaktisch höherwertige API bieten. Das Ziel ist dass der Endanwender kein tieferes Verständnis für die Anwendung braucht. Der Endanwender soll beispielsweise Memory Mapped Files problemlos verwenden können, ohne jedoch deren Ablauf verstehen zu müssen.

Auch wenn Boost.Interprocess als Hauptvergleichspunkt dient, werden hier nun alle Vergleiche durchgenommen.

2.1.1 Boost.Interprocess

Das Ziel von Boost.Interprocess ist eine vereinfachte Plattformübergreifende Softwarelösung for MPC und IPC. Es bietet einige Features wie Shared memory oder Memory-mapped files an. Die Bibliothek ist mächtig und sehr stabil und befindet sich nahe am C++ Standard, was schließlich der Grund ist warum die Boost.Interprocess als Referenz ausgewählt wurde. Eines der Hauptprobleme an Boost.Interprocess ist nicht nur seine schwer zugängliche Syntax, sondern auch der Overhead den diese mitbringt. Ziel von *libipc++* ist es eine ähnlich mächtige API mit geringeren Overhead und einer schöneren Syntax zu realisieren.

2.1.2 OpenMP

Neben Boost.Interprocess hat sich die OpenMP als quasi Standard durchgesetzt. Sie wird von allen großen Herstellern und OS unterstützt. Jedoch versucht OpenMP Parallelität durch den C Präprozessor zu realisieren. Diese Art von Design wird als überholt und veraltet erachtet. Aus diesem Grund wurde OpenMP nicht als Referenz zu diesen Projekt gewählt.

2.2 Produktfunktion

libipc++ soll wie schon erwähnt eine MPC und IPC bieten, hierzu wird auf Systemressourcen zurückgegriffen. Im folgenden findet sich ein kleiner Überblick über die Funktionalität der Software.

2.2.1 Prozess

Als Grundfunktion was IPC überhaupt erst notwendig macht, ist es einen neuen Prozess innerhalb eines Programms zur Laufzeit zu starten. Diese Funktionalität ist sehr stark vom verwendeten OS abhängig. In diesem wird daher lediglich das allgemeine Konzept eines Kindprozesses innerhalb eines Programms besprochen.

Wenn ein ausgeführtes Programm einen neuen Prozess zur Laufzeit startet, dann läuft dieser als Kindprozess des ausführenden Prozesses. Der neue erstellte Prozess besitzt dann eine eigene Laufzeitumgebung was bedeutet dass er einen eigenen virtuellen Speicherbereich vom OS zugewiesen bekommt. Das Ziel eines Prozesses ist es jedoch meist eine Berechnung oder Anweisung parallel durchzuführen, was bedeutet dass es zumindest ein relevantes Ergebnis gibt. Da der Prozess jedoch einen eigenen virtuellen Speicherbereich besitzt, muss der Speicheraustausch über andere Mechanismen stattfinden.

2.2.2 Pipes

Eine weitverbreitete Variante der Kommunikation sind Pipes, welche durch Unix populär wurden und inzwischen von den meisten Betriebssystemen unterstützt werden. Pipes übertragen sequentiell einzelne Bytes. Jedem Prozess sind immer zwei Datenströme zugeordnet, einer für die Ausgabe und einer für die Eingabe. Ein Ausgabestrom eines Prozesses kann anschließend direkt mit dem Eingabestrom eines anderen Prozesses verbunden werden.
Eine Erweiterung dieses Konzeptes stellen **Named Pipes** dar: Diese sind nicht einem Prozess fest zugeordnet, sondern können genau wie Dateien geschrieben und gelesen werden. Sie existieren als Objekte im Dateisystem mit einem Namen, und bieten so eine sehr flexible Schnittstelle für die Kommunikation mit einem Prozess.

2.2.3 Message Queues

Message Queues sind wesentlich weniger Verbreitet als Pipes und erlauben es Nachrichten in einzelnen Paketen zu versenden. Diese Kommunikationsform hat den Vorteil dass Nachrichten eine Priorität zugewiesen werden kann, dies erlaubt es bestimmte Nachrichten vor anderen aus dem Queue abzurufen.

2.2.4 Memory-mapped File

Memory-mapped Files erlauben es eine Datei direkt in den Speicher zu kopieren. Besonders bei sehr großen Dateien kann sich dies positiv auf die I/O Performance auswirken.

2.2.5 Semaphore

Semaphoren ermöglichen den Zugriff auf Shared Memory, Dateien oder andere Ressourcen zu kontrollieren.

2.2.6 File Locking

File Locking ist eine vereinfachte Variante der Zugriffskontrolle speziell für Dateien.

2.2.7 Shared Memory

Shared Memory erlaubt mehreren Prozessen Zugriff auf den selben Speicherbereich. Bei der Verwendung von Shared Memory ist zu beachten dass es zu Concurrency Problemen kommen kann. Daher ist die Verwendung einer Semaphore zur Zugriffskontrolle empfohlen.

2.3 Benutzermerkmale

Da es sich bei der *libipc++* um eine Programmbibliothek handelt, werden vom Endanwender gewisse Kenntnisse vorausgesetzt, im folgenden werden diese kurz beschrieben.

2.3.1 C++

Die Bibliothek wird in `C++` implementiert, was bedeutet dass der Endanwender zumindest fortgeschrittene Kenntnisse in `C++` aufweisen sollte. Als Mindestvoraussetzung gilt die Version `C++11`. Verwendet werden mehrere Sprachkonzepte von `C++`, darunter auch Templates.

2.3.2 Design Patterns

Ebenfalls wird viel Wert auf Design Patterns gelegt. Wichtig ist dass die Bibliothek leicht erweiterbar und schnell auf Veränderungen reagieren kann. Der Endanwender braucht jedoch nicht unbedingt ein umfangreiches Wissen über Design Patterns, es genügt ein hohes Verständniss über Object oriented programming (OOP).

2.4 Einschränkungen

Da *libipc++* als cross-platform Bibliothek entwickelt wird, muss als das Codetool CMake verwendet werden. Unterstützt werden alle gängigen IDE, sowie UNIX makefile.

2.5 Annahme und Abhängigkeiten

Die *libipc++* wird als Studentenprojekt an der FH Technikum Wien gemacht. Daraus ergibt sich dass für das Projekt nicht der volle Zeitaufwand aufgebracht werden kann wie in einen eigentlichen Softwareprojekt, da es nebenbei noch andere Projekte und Lehrveranstaltung (LVA) gibt.

2.5.1 Message Queues

Da die Microsoft Message Queues (MSMQ) Bibliothek wesentlich komplexer ist als zuvor angenommen. Was dazu führen kann dass das Feature unter Win32 nicht vollständig implementiert wird.

2.5.2 Betriebssystem

Entwickelt wird *libipc++* für Windows und Linux, bei Linux speziell ein OpenSUSE Derivat. Der Code wird entsprechend auf diesen beiden OS getestet werden.

2.5.3 Nebenberuflich

Da eine nebenberufliche Tätigkeit von einen Teammitglied nachgegangen wird, kann es zu einigen Verzögerungen innerhalb des Projektflusses kommen. Es wird jedoch darauf geachtet dass diese Tätigkeit keinen zu großen Einfluss auf das Projekt hat.

2.5.4 Andere Lehrveranstaltungen

Das Projekt wird neben anderen LVAs im Studiengang Bachelor Informatik an der FH Technikum Wien gemacht. Dies kann dann natürlich aufgrund von Überkreuzungen zu einer Verzögerung des Projekts führen.

2.6 Aufteilung der Anforderungen

Anforderungen die nicht zum jetzigen Release miteinbezogen werden können sind

- Frontend der Bibliothek,
- Ausbau von NamedPipes,
- serialisieren von Datenobjekten,
- umfangreiches Profiling,
- Benchmark testing und
- Performancetests im allgemeinen.

Diese Features sind für zukünftige Versionen gedacht. Falls es sich jedoch ausgehen sollte diese Features im kommenden Release zu implementieren, wird die

Spezifikation entsprechen aktualisiert. Als nächstes eine Liste von problematische Features die vielleicht nicht realisiert werden können wenn der tatsächlichen Aufwand höher ist als der geschätzte ist.

- MessageQueues (Win32)

Kapitel 3

Spezifische Anforderungen

Im Gegensatz zum zweiten Kapitel, wo nur allgemeine Anforderungen und Beschreibungen vorgenommen wurden, werden hier nun die Anforderungen im Detail spezifiziert. Beginnend mit den funktionalen Anforderungen die Themen wie die Bibliotheksfunktionen beinhalten zu den nicht-funktionalen Anforderungen die Themen wie

- Zuverlässigkeit,
- Benutzbarkeit,
- Wartbarkeit,
- Flexibilität,
- Skalierbarkeit und
- Sicherheit

im Detail beschreiben. Danach werden noch externe APIs und andere verwendete externe Produkte beschrieben. Zum Schluss gibt es noch einen Überblick über die geforderte Performance, den zu erwartenden Overhead und einen Vergleich mit anderen Produkten der selben Kategorie.

3.1 Funktionale Anforderungen

Die *libipc++* teilt sich im wesentlichen in zwei Blöcke ein. Den Plattformspezifischen Block der dazu dient die OS API als ein einheitliches Interface zu adaptieren. Dieser Block wird als Backend bezeichnet. Und den zweiten Block der dann die abstrahierte OS API zu einer schönen high level API adaptiert. Dieser Block wird als Frontend bezeichnet.

Die Beschreibung der Win32 Argumente wird in Englisch gehalten, der Grund hierfür ist die ebenfalls englische Ausführung auf der MSDN (Microsoft Developer Network).

3.1.1 Backend

Dadurch dass die *libipc++* auf unterschiedlichen OS funktionieren soll, diese jedoch kein einheitliches System von IPC besitzen, ist es notwendig die zur Verfügung stehende OS API zu einer einheitlichen abstrakten API zusammenzufassen. Im nachfolgenden werden die OS spezifischen Funktionen genauer erläutert. Angefangen wird mit der Erzeugung eines neuen Prozesses. Es sei noch angemerkt dass immer zuerst die Win32 Funktion erläutert wird, danach die UNIX Funktion. Der Grund ist dass die UNIX Funktionen meist wohlbekannt sind, jedoch die Win32 Funktionen, dadurch dass sie seltener verwendet werden, eher unbekannt sind.

Sämtliche hier präsentierten Funktionen können auch selber in der entsprechenden Dokumentation nachgelesen werden. Der Verweis befindet sich unter der Sektion externe Ressourcen.

Prozess

Listing 3.1: class Process

```
class IPC_API Process : public ReferenceType {
  public:
    Process();
    Process(Process&& process);
    virtual ~Process();

    int ExitCode() const;
    ProcessHandle Handle() const;
    void Kill() const;
    bool Valid() const;
    void Wait() const;

    template<typename Rep, typename Period>
    bool WaitFor(const std::chrono::duration<Rep, Period
        >& timeoutDuration) const;

    template <typename Clock, typename Duration>
    bool WaitUntil(const std::chrono::time_point<Clock,
        Duration>& timoutTime) const;

    Process& operator=(Process&& process);
    bool operator==(const Process& process) const;
    bool operator!=(const Process& process) const;

  private:
};
```

3.1.2 Frontend

Das Frontend stellt die high level API dar. Das Frontend ist anders als das Backend nicht von der OS API abhängig, sondern von der Abstrahierten API die unter Backend beschrieben wurde.

Dieses Feature wird erst im nächsten Release implementiert.

3.2 Nicht-Funktionale Anforderungen

3.2.1 Versionierung

0.1	Abstrakte API
0.2	Process
0.3	AnonymousPipe
0.4	NamedPipe
0.5	MessageQueue
0.6	Semaphore
0.7	MemoryMap, SharedMemory
0.8	FileLocking
0.9	Profiling Windows und Linux
1.0	Release

Die erste Version bietet noch keine Features an. Es wird die API von *libipc++* implementiert und auf Konsistenz getestet. Dabei werden alle notwendigen und definierten Klassen beschrieben, deren Methoden und Felder dokumentiert sowie Utility-Funktionalität beschrieben. Es ist wichtig dass das Design eine eigene Version erhält, da dadurch erst mit der Implementierung begonnen werden kann wenn das Design voll funktionsfähig und getestet ist. Folgende Klassen müssen definiert und getestet sein bevor die Version abgeschlossen wird.

Jede Version nach der Version 0.1 implementiert dann die entsprechende Klasse für die unterstützten Plattformen. Eine Klasse gilt erst als fertig wenn

- sie vollständig implementiert ist,
- sie ausreichend getestet ist,
- sie umfangreich dokumentiert ist
- und best practice als auch Beispielcode für jede Methode existiert.

Eine Ausnahme stellt hier die Version 0.9 dar. Diese Version dient lediglich dazu den bereits existierenden Code zu optimieren. Primär für die Geschwindigkeit. Wie bereits erwähnt, kann es zu Verzögerungen bei dieser Version kommen.

3.3 Externe Schnittstellen

Da die *libipc++* keine Abhängigkeiten von anderen Programmbibliotheken besitzt, gibt es als externe Schnittstelle nur die von OS angebotene API für MPC und IPC. Eine Ausnahme stellt hier der Testingbereich dar. Dieser hat als Abhängigkeit das gTest Framework, gTest wird jedoch mit der Repo mitgeliefert um eine selbstständige Kompilierung samt Testing zu ermöglichen. Im Folgenden werden die Schnittstellen der OS API kurz beschrieben.

3.3.1 Win32

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Multiprocessing

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Pipes

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Message Queues

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Memory Mapped Files

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Shared Memory

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Semaphore

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

3.3.2 UNIX

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

3.4 Anforderungen an Performance

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

3.5 Qualitätsanforderungen

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.