# 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 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

| | |
|---|---|
| Kanban | Form von Projektmanagement |
| SRS | Software requirement specification |
| MPC | Multi processing |
| IPC | Interprocess communication |
| OS | Operating system |
| API | Application programming interface |
| FIFO | First In First Out |
| OOP | Object oriented programming |
| IDE | Integrated development environment |
| LVA | Lehrveranstaltung |
| MSDN | Microsoft Developer Network |
| MSMQ | Microsoft Message Queues |

## 1.4 Verweise auf sonstige Ressourcen oder Quellen

| | |
|---|---|
| FH Technikum Wien | `http://www.technikum-wien.at/` |
| SRS | `https://de.wikipedia.org/wiki/Software_Requirements_Specification` |
| 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/ms684852%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 |

# Kapitel 2

# Allgemeine Beschreibung

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

Die Bibliothek wird auf allen gängigen und modernen OS (Operating system) 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

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.

### 2.2.3 Message Queus

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.

### 2.2.4 Memory-mapped File

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.

### 2.2.5 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.

### 2.2.6 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.

## 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 OOP (Object oriented programming).

## 2.4 Einschränkungen

Da *libipc++* als cross-platform Bibliothek entwickelt wird, muss als das Codetool CMake verwendet werden. Unterstützt werden alle gängigen IDEs (Integrated development environment), 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 LVAs (Lehrveranstaltungen) gibt.

### 2.5.1 Message Queues

Da die MSMQ (Microsoft Message Queues) 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 (Lehrveranstaltungen) 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, dann 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 Aufwand 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

Ein Prozess wird im wesentlichen durch den Aufruf einer einzelnen Funktion erzeugt.

Listing 3.1: CreateProcess

```
BOOL WINAPI CreateProcess(
  _In_opt_    LPCTSTR lpApplicationName,
  _Inout_opt_ LPTSTR lpCommandLine,
  _In_opt_    LPSECURITY_ATTRIBUTES lpProcessAttributes,
  _In_opt_    LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_        BOOL bInheritHandles,
  _In_        DWORD dwCreationFlags,
  _In_opt_    LPVOID lpEnvironment,
  _In_opt_    LPCTSTR lpCurrentDirectory,
  _In_        LPSTARTUPINFO lpStartupInfo,
  _Out_       LPPROCESS_INFORMATION lpProcessInformation
);
```

*lpApplicationName* [in, optional]

> The name of the module to be executed. This module can be a Windows-based application. It can be some other type of module (for example, MS-DOS or OS/2) if the appropriate subsystem is available on the local computer.
>
> The string can specify the full path and file name of the module to execute or it can specify a partial name. In the case of a partial name, the function uses the current drive and current directory to complete the specification. The function will not use the search path. This parameter must include the file name extension; no default extension is assumed.
>
> The *lpApplicationName* parameter can be **NULL**. In that case, the module name must be the first white space–delimited token in the *lpCommandLine* string. If you are using a long file name that contains a space, use quoted strings to indicate where the file name ends and the arguments

begin; otherwise, the file name is ambiguous. For example, consider the string "c:\program files\sub dir\program name". This string can be interpreted in a number of ways. The system tries to interpret the possibilities in the following order:

**c:\program.exe** files\sub dir\program name
**c:\program files\sub.exe** dir\program name
**c:\program files\sub dir\program.exe** name
**c:\program files\sub dir\program name.exe**

If the executable module is a 16-bit application, *lpApplicationName* should be **NULL**, and the string pointed to by *lpCommandLine* should specify the executable module as well as its arguments.

To run a batch file, you must start the command interpreter; set *lpApplicationName* to cmd.exe and set *lpCommandLine* to the following arguments: /c plus the name of the batch file.

*lpCommandLine* [in, out, optional]

The command line to be executed. The maximum length of this string is 32,768 characters, including the Unicode terminating null character. If *lpApplicationName* is **NULL**, the module name portion of *lpCommandLine* is limited to **MAX_PATH** characters.

The Unicode version of this function, CreateProcessW, can modify the contents of this string. Therefore, this parameter cannot be a pointer to read-only memory (such as a const variable or a literal string). If this parameter is a constant string, the function may cause an access violation.

The *lpCommandLine* parameter can be **NULL**. In that case, the function uses the string pointed to by *lpApplicationName* as the command line.

If both *lpApplicationName* and *lpCommandLine* are non-**NULL**, the null-terminated string pointed to by *lpApplicationName* specifies the module to execute, and the null-terminated string pointed to by *lpCommandLine* specifies the command line. The new process can use GetCommandLine to retrieve the entire command line. Console processes written in C can use the argc and argv arguments to parse the command line. Because argv[0] is the module name, C programmers generally repeat the module name as the first token in the command line.

If *lpApplicationName* is **NULL**, the first white space–delimited token of the command line specifies the module name. If you are using a long file name that contains a space, use quoted strings to indicate where the file name ends and the arguments begin (see the explanation for the *lpApplicationName* parameter). If the file name does not contain an extension, .exe is appended. Therefore, if the file name extension is .com, this parameter must include the .com extension. If the file name ends in a period (.) with no extension, or if the file name contains a path, .exe is not appended. If the file name does not contain a directory path, the system searches for the executable file in the following sequence:

1. The directory from which the application loaded.

2. The current directory for the parent process.
3. The 32-bit Windows system directory. Use the GetSystemDirectory function to get the path of this directory.
4. The 16-bit Windows system directory. There is no function that obtains the path of this directory, but it is searched. The name of this directory is System.
5. The Windows directory. Use the GetWindowsDirectory function to get the path of this directory.
6. The directories that are listed in the PATH environment variable. Note that this function does not search the per-application path specified by the App Paths registry key. To include this per-application path in the search sequence, use the ShellExecute function.

The system adds a terminating null character to the command-line string to separate the file name from the arguments. This divides the original string into two strings for internal processing.

*lpProcessAttributes* [in, optional]

A pointer to a **SECURITY_ATTRIBUTES** structure that determines whether the returned handle to the new process object can be inherited by child processes. If *lpProcessAttributes* is **NULL**, the handle cannot be inherited.

The *lpSecurityDescriptor* member of the structure specifies a security descriptor for the new process. If *lpProcessAttributes* is **NULL** or *lpSecurityDescriptor* is **NULL**, the process gets a default security descriptor. The ACLs in the default security descriptor for a process come from the primary token of the creator.

**Windows XP:** The ACLs in the default security descriptor for a process come from the primary or impersonation token of the creator. This behavior changed with Windows XP with SP2 and Windows Server 2003.

*lpThreadAttributes* [in, optional]

A pointer to a **SECURITY_ATTRIBUTES** structure that determines whether the returned handle to the new thread object can be inherited by child processes. If *lpThreadAttributes* is **NULL**, the handle cannot be inherited.

The *lpSecurityDescriptor* member of the structure specifies a security descriptor for the main thread. If *lpThreadAttributes* is **NULL** or *lpSecurityDescriptor* is **NULL**, the thread gets a default security descriptor. The ACLs in the default security descriptor for a thread come from the process token.

**Windows XP:** The ACLs in the default security descriptor for a thread come from the primary or impersonation token of the creator. This behavior changed with Windows XP with SP2 and Windows Server 2003.

*bInheritHandles* [in]

If this parameter TRUE, each inheritable handle in the calling process is inherited by the new process. If the parameter is FALSE, the handles are not inherited. Note that inherited handles have the same value and access rights as the original handles.

*dwCreationFlags* [in]

The flags that control the priority class and the creation of the process. For a list of values, see Process Creation Flags.

This parameter also controls the new process's priority class, which is used to determine the scheduling priorities of the process's threads. For a list of values, see **GetPriorityClass**. If none of the priority class flags is specified, the priority class defaults to **NORMAL_PRIORITY_CLASS** unless the priority class of the creating process is **IDLE_PRIORITY_CLASS** or **BELOW_NORMAL_PRIORITY_CLASS**. In this case, the child process receives the default priority class of the calling process.

*lpEnvironment* [in, optional]

A pointer to the environment block for the new process. If this parameter is **NULL**, the new process uses the environment of the calling process.

An environment block consists of a null-terminated block of null-terminated strings. Each string is in the following form:

name=value\0

Because the equal sign is used as a separator, it must not be used in the name of an environment variable.

An environment block can contain either Unicode or ANSI characters. If the environment block pointed to by lpEnvironment contains Unicode characters, be sure that *dwCreationFlags* includes **CREATE_UNICODE_ENVIRONMENT**. If this parameter is **NULL** and the environment block of the parent process contains Unicode characters, you must also ensure that *dwCreationFlags* includes **CREATE_UNICODE_ENVIRONMENT**.

The ANSI version of this function, *CreateProcessA* fails if the total size of the environment block for the process exceeds 32,767 characters.

Note that an ANSI environment block is terminated by two zero bytes: one for the last string, one more to terminate the block. A Unicode environment block is terminated by four zero bytes: two for the last string, two more to terminate the block.

*lpCurrentDirectory* [in, optional]

The full path to the current directory for the process. The string can also specify a UNC path.

If this parameter is **NULL**, the new process will have the same current drive and directory as the calling process. (This feature is provided primarily for shells that need to start an application and specify its initial drive and working directory.)

*lpStartupInfo* [in]

A pointer to a **STARTUPINFO** or **STARTUPINFOEX** structure.

To set extended attributes, use a **STARTUPINFOEX** structure and specify **EXTENDED_STARTUPINFO_PRESENT** in the *dwCreationFlags* parameter.

Handles in **STARTUPINFO** or **STARTUPINFOEX** must be closed with **CloseHandle** when they are no longer needed.

Important The caller is responsible for ensuring that the standard handle fields in **STARTUPINFO** contain valid handle values. These fields are copied unchanged to the child process without validation, even when the *dwFlags* member specifies **STARTF_USESTDHANDLES**. Incorrect values can cause the child process to misbehave or crash. Use the Application Verifier runtime verification tool to detect invalid handles.

*lpProcessInformation* [out]

A pointer to a **PROCESS_INFORMATION** structure that receives identification information about the new process.

Handles in **PROCESS_INFORMATION** must be closed with CloseHandle when they are no longer needed.

**Return value**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Note that the function returns before the process has finished initialization. If a required DLL cannot be located or fails to initialize, the process is terminated. To get the termination status of a process, call **GetExitCodeProcess**.

Die weiteren Funktionen für Prozessmanagement können auf der MSDN nachgelesen werden. Als nächstes folgt das UNIX Pendant.

Listing 3.2: fork

```
pid_t fork(void);
```

Auch hier können die restlichen Prozessmanagementfunktionen in der offiziellen Dokumentation nachgelesen werden. Als nächstes wird nun die zu adaptierende Klasse beschrieben.

Listing 3.3: 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 so-

wie 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 dann 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

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.