

MAKALAH

Defer dan Exit

Disusun untuk memenuhi tugas UAS



Dosen Pengampu:
Muhammad Husen, M.kom

Disusun oleh:
Khoirul Imam Fazri 2025010016

PROGRAM STUDI
REKAYASA PERANGKAT LUNAK
POLITEKNIK BALEKAMBANG JEPARA
2026

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadiran Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga makalah dengan judul “Defer dan Exit” ini dapat diselesaikan dengan baik. Makalah ini disusun untuk memenuhi tugas mata kuliah pemrograman dan sebagai sarana untuk memperdalam pemahaman mengenai mekanisme pengelolaan sumber daya serta penghentian program dalam bahasa pemrograman modern.

Defer dan exit merupakan dua konsep yang sering dijumpai dalam pengembangan perangkat lunak, khususnya dalam bahasa pemrograman seperti Go, C, Python, dan Java. Pemahaman yang baik terhadap kedua konsep ini akan membantu programmer dalam menulis kode yang lebih rapi, aman, dan bebas dari kebocoran sumber daya (resource leak).

Penulis menyadari bahwa makalah ini masih jauh dari kata sempurna. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan demi perbaikan di masa yang akan datang. Semoga makalah ini dapat memberikan manfaat dan menambah wawasan bagi pembaca, khususnya dalam bidang pemrograman komputer.

Penulis

DAFTAR ISI

KATA PENGANTAR	i
DAFTAR ISI.....	ii
BAB I PENDAHULUAN	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah	2
1.3 Tujuan Penulisan	2
BAB II PEMBAHASAN	4
2.1 Pengertian Defer.....	4
2.2 Sintaks dan Cara Kerja Defer	4
2.3 Aturan Eksekusi Defer (LIFO)	5
2.4 Contoh Penerapan Defer	6
2.5 Pengertian Exit.....	6
2.6 Jenis-Jenis Status Keluar (Exit Code).....	7
2.7 Cara Kerja Fungsi Exit	7
2.8 Contoh Penerapan Exit.....	8
2.9 Interaksi antara Defer dan Exit.....	8
2.10 Perbandingan Defer dan Exit	9
2.11 Dampak dan Praktik Terbaik.....	10
BAB III PENUTUP	11
3.1 Kesimpulan	11
3.2 Saran	11
SOAL LATIHAN	13

DAFTAR PUSTAKA.....	14
LAMPIRAN.....	15

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam dunia pemrograman, pengelolaan sumber daya seperti berkas (file), koneksi jaringan, koneksi basis data, dan kunci (lock) merupakan hal yang sangat penting untuk diperhatikan. Kesalahan dalam mengelola sumber daya tersebut dapat menyebabkan kebocoran sumber daya (resource leak), yang pada akhirnya dapat menurunkan kinerja aplikasi bahkan menyebabkan program berhenti bekerja secara tidak normal. Untuk mengatasi permasalahan tersebut, banyak bahasa pemrograman modern menyediakan mekanisme khusus yang memungkinkan seorang programmer menjadwalkan proses pembersihan sumber daya secara otomatis.

Salah satu mekanisme yang banyak digunakan, terutama pada bahasa pemrograman Go, adalah pernyataan defer. Pernyataan ini memungkinkan sebuah pemanggilan fungsi dijadwalkan untuk dijalankan setelah fungsi yang berjalan saat ini selesai dieksekusi, tanpa memandang apakah fungsi tersebut berakhir secara normal maupun karena terjadinya kesalahan (error atau panic). Konsep serupa juga dapat ditemukan pada bahasa pemrograman lain dalam bentuk yang berbeda, misalnya blok try-finally pada Java dan Python, maupun konsep RAII (Resource Acquisition Is Initialization) pada bahasa C++.

Di sisi lain, terdapat pula mekanisme exit yang berfungsi untuk menghentikan eksekusi program secara langsung dan segera, baik karena program telah menyelesaikan tugasnya maupun karena ditemukan kondisi kesalahan yang membuat program tidak dapat melanjutkan proses. Fungsi

exit umumnya juga mengembalikan sebuah kode status (exit code) kepada sistem operasi yang dapat digunakan untuk menandakan apakah program berhasil dijalankan atau mengalami kegagalan.

Yang menarik untuk dikaji lebih lanjut adalah bagaimana interaksi antara defer dan exit, sebab pada beberapa bahasa pemrograman, khususnya Go, pemanggilan fungsi keluar seperti `os.Exit()` tidak akan menjalankan proses defer yang telah dijadwalkan sebelumnya. Hal ini dapat menimbulkan kesalahan logika program apabila programmer tidak memahami perbedaan mendasar antara keduanya. Oleh karena itu, penulis tertarik untuk membahas topik ini secara lebih mendalam dalam makalah yang berjudul “Defer dan Exit”.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah dalam makalah ini adalah sebagai berikut:

1. Apa yang dimaksud dengan defer dalam pemrograman?
2. Bagaimana cara kerja dan aturan eksekusi pernyataan defer?
3. Apa yang dimaksud dengan exit dalam pemrograman?
4. Bagaimana cara kerja fungsi exit pada sebuah program?
5. Bagaimana pengaruh pemanggilan exit terhadap defer yang telah dijadwalkan sebelumnya?
6. Apa perbedaan mendasar antara defer dan exit, serta bagaimana praktik terbaik dalam penggunaannya?

1.3 Tujuan Penulisan

Berdasarkan rumusan masalah di atas, maka tujuan penulisan makalah ini adalah sebagai berikut:

1. Untuk mengetahui dan memahami pengertian defer dalam pemrograman.
2. Untuk memahami cara kerja dan aturan eksekusi pernyataan defer.
3. Untuk mengetahui dan memahami pengertian exit dalam pemrograman.
4. Untuk memahami cara kerja fungsi exit pada sebuah program.
5. Untuk mengetahui pengaruh pemanggilan exit terhadap defer yang telah dijadwalkan.
6. Untuk mengetahui perbedaan mendasar antara defer dan exit beserta praktik terbaik penggunaannya.

BAB II

PEMBAHASAN

2.1 Pengertian Defer

Defer adalah sebuah pernyataan dalam bahasa pemrograman yang digunakan untuk menjadwalkan pemanggilan suatu fungsi agar dijalankan setelah fungsi yang sedang berjalan (fungsi pemanggil) selesai dieksekusi, tanpa memandang apakah fungsi tersebut berakhir secara normal melalui pernyataan return, maupun berakhir karena terjadinya kondisi kesalahan (panic). Konsep defer pertama kali dipopulerkan oleh bahasa pemrograman Go (Golang) dan menjadi salah satu ciri khas dari bahasa tersebut.

Tujuan utama dari penggunaan defer adalah untuk memastikan bahwa proses pembersihan sumber daya, seperti menutup berkas, menutup koneksi jaringan, melepas kunci (mutex), maupun mencatat log, akan selalu dijalankan meskipun terjadi kesalahan di tengah proses. Dengan demikian, penggunaan defer dapat membuat kode program menjadi lebih ringkas, mudah dibaca, dan lebih kecil kemungkinannya mengalami kebocoran sumber daya dibandingkan apabila proses pembersihan ditulis secara manual di setiap kemungkinan jalur keluar fungsi.

2.2 Sintaks dan Cara Kerja Defer

Secara sintaks, pernyataan defer pada bahasa Go dituliskan dengan menambahkan kata kunci defer sebelum pemanggilan fungsi. Berikut adalah contoh penerapannya:

```
func bacaBerkas() {  
    berkas, _ := os.Open("data.txt")
```



```

defer berkas.Close()

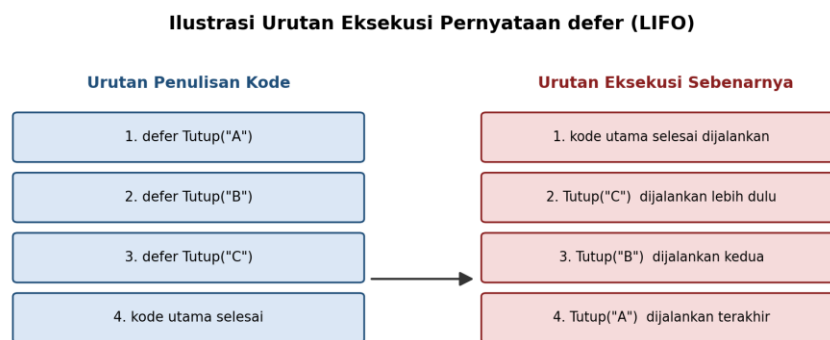
// proses pembacaan berkas
// ...
} // berkas.Close() otomatis dijalankan di sini

```

Terdapat satu hal penting yang perlu diperhatikan, yaitu argumen dari fungsi yang di-defer akan dievaluasi pada saat pernyataan defer dijalankan (bukan pada saat fungsi tersebut benar-benar dipanggil di akhir). Namun, eksekusi fungsinya sendiri baru akan dilakukan setelah fungsi pemanggil selesai. Pemahaman ini penting agar programmer tidak salah menduga nilai variabel yang digunakan di dalam fungsi yang di-defer.

2.3 Aturan Eksekusi Defer (LIFO)

Apabila di dalam satu fungsi terdapat lebih dari satu pernyataan defer, maka urutan eksekusinya tidak mengikuti urutan penulisan, melainkan mengikuti prinsip Last In First Out (LIFO), yaitu fungsi yang di-defer paling akhir akan dijalankan paling awal. Prinsip ini serupa dengan cara kerja struktur data stack (tumpukan).



Fungsi defer yang dipanggil paling akhir akan dieksekusi paling awal saat fungsi utama berakhir (prinsip Last In First Out / LIFO)

Gambar 1. Ilustrasi urutan eksekusi pernyataan defer berdasarkan prinsip LIFO

Sebagai contoh, apabila di dalam sebuah fungsi terdapat tiga pernyataan defer yang dituliskan secara berurutan untuk menutup sumber

daya A, B, dan C, maka pada saat fungsi tersebut berakhir, urutan eksekusinya adalah C terlebih dahulu, kemudian B, dan terakhir A. Pemahaman terhadap prinsip LIFO ini sangat penting terutama ketika urutan pembersihan sumber daya memiliki ketergantungan satu sama lain.

2.4 Contoh Penerapan Defer dalam Berbagai Kasus

Penggunaan defer tidak terbatas hanya pada penutupan berkas, tetapi dapat diterapkan pada berbagai kebutuhan lain, di antaranya:

- Menutup berkas atau koneksi basis data setelah selesai digunakan.
- Melepas kunci (`mutex.Unlock()`) pada proses pemrograman konkuren agar tidak terjadi kebuntuan (deadlock).
- Mencatat waktu mulai dan waktu selesai eksekusi suatu fungsi untuk keperluan pengukuran kinerja (profiling).
- Menangani dan memulihkan kondisi panic melalui fungsi `recover()` agar program tidak berhenti secara tiba-tiba.
- Menjamin nilai kembalian (return value) suatu fungsi diperbarui sebelum fungsi benar-benar berakhir, melalui teknik named return value.

2.5 Pengertian Exit

Exit adalah sebuah mekanisme atau fungsi yang digunakan untuk menghentikan eksekusi suatu program secara langsung dan segera. Ketika fungsi exit dipanggil, program tidak akan melanjutkan baris kode berikutnya, melainkan langsung mengembalikan kendali kepada sistem operasi disertai dengan sebuah kode status yang disebut exit code. Hampir seluruh bahasa pemrograman menyediakan mekanisme ini, meskipun dengan penamaan dan perilaku yang sedikit berbeda, misalnya fungsi `exit()` pada bahasa C, `sys.exit()` pada Python, `os.Exit()` pada Go, dan `System.exit()` pada Java.

2.6 Jenis-Jenis Status Keluar (Exit Code)

Exit code merupakan sebuah bilangan bulat yang menandakan status akhir dari sebuah program pada saat berakhir. Secara konvensi, sebagian besar sistem operasi menetapkan aturan sebagai berikut:

- Exit code 0 (nol) menandakan bahwa program berhasil dijalankan tanpa terjadi kesalahan.
- Exit code selain 0 (bernilai 1 sampai dengan 255) menandakan bahwa program berakhir karena suatu kondisi kesalahan tertentu.

Konvensi ini memungkinkan program lain, skrip otomatisasi, maupun sistem operasi untuk mengetahui apakah suatu program telah berjalan dengan sukses atau mengalami kegagalan, tanpa perlu membaca keluaran (output) program secara langsung.

2.7 Cara Kerja Fungsi Exit

Ketika fungsi exit dipanggil, program akan langsung dihentikan pada saat itu juga. Seluruh baris kode setelah pemanggilan exit tidak akan pernah dieksekusi. Berikut adalah contoh sederhana penerapan fungsi exit pada bahasa Go:

```
func proses() {  
    fmt.Println("Memulai proses")  
    if terjadiKesalahan() {  
        os.Exit(1) // program langsung berhenti di sini  
    }  
    fmt.Println("Baris ini tidak akan pernah tercetak")  
}
```

Perlu diketahui bahwa perilaku fungsi exit terhadap mekanisme pembersihan sumber daya berbeda-beda antar bahasa pemrograman. Pada bahasa Go, pemanggilan `os.Exit()` akan menghentikan program secara langsung tanpa menjalankan defer yang telah dijadwalkan. Sementara itu, pada bahasa Python, pemanggilan `sys.exit()` sebenarnya bekerja dengan cara memicu sebuah exception bernama `SystemExit`,

sehingga blok `finally` yang sedang aktif tetap akan dijalankan sebelum program benar-benar berakhir. Adapun fungsi `os._exit()` pada Python bersifat lebih ekstrem karena menghentikan program tanpa menjalankan proses pembersihan apa pun.

2.8 Contoh Penerapan Exit

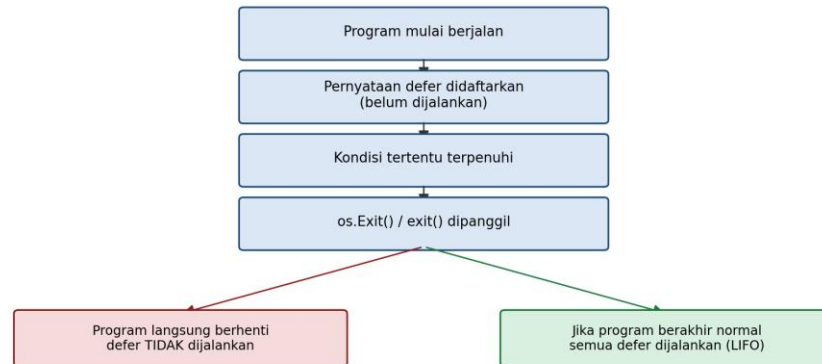
Fungsi `exit` umumnya digunakan pada situasi berikut:

- Menghentikan program ketika terjadi kesalahan fatal yang tidak memungkinkan program untuk melanjutkan proses, misalnya berkas konfigurasi tidak ditemukan.
- Memberikan kode status tertentu kepada skrip otomatisasi atau sistem continuous integration (CI/CD) yang menjalankan program tersebut.
- Menghentikan proses anak (child process) dalam pemrograman berbasis command-line.
- Keluar dari program utama setelah seluruh proses inti selesai dijalankan.

2.9 Interaksi antara Defer dan Exit

Bagian ini merupakan inti pembahasan dari makalah ini, yaitu bagaimana `defer` dan `exit` saling berinteraksi. Pada bahasa Go, apabila fungsi `os.Exit()` dipanggil, seluruh pernyataan `defer` yang telah dijadwalkan sebelumnya, baik di dalam fungsi tersebut maupun di fungsi lain yang berada di atasnya dalam tumpukan pemanggilan (call stack), tidak akan pernah dijalankan. Hal ini berbeda dengan kondisi ketika sebuah fungsi berakhir secara normal melalui pernyataan `return`, di mana seluruh `defer` yang terdaftar akan tetap dijalankan sesuai urutan LIFO sebelum fungsi benar-benar berakhir.

Ilustrasi Efek Pemanggilan exit() terhadap defer



Perbedaan utama: exit() menghentikan program secara langsung tanpa membersihkan sumber daya via defer, sedangkan akhir normal fungsi akan menjalankan seluruh defer yang terdaftar.

Gambar 2. Ilustrasi efek pemanggilan exit() terhadap defer yang telah dijadwalkan

Kondisi ini menjadikan penggunaan `os.Exit()` perlu dilakukan dengan sangat hati-hati, terutama pada program yang mengelola sumber daya penting seperti berkas terbuka, koneksi basis data, maupun kunci yang sedang dipegang oleh suatu proses. Apabila `os.Exit()` dipanggil sebelum sumber daya tersebut dibersihkan secara manual, maka sumber daya tersebut berpotensi tidak pernah dilepaskan dengan benar, meskipun sistem operasi pada umumnya akan tetap membersihkan sebagian sumber daya tingkat rendah setelah proses benar-benar berakhir.

2.10 Perbandingan Defer dan Exit

Untuk memperjelas perbedaan antara defer dan exit, berikut disajikan tabel perbandingan keduanya:

Aspek	Defer	
Fungsi utama	Menjadwalkan pembersihan sumber daya	Meng
Waktu eksekusi	Dijalankan saat fungsi pemanggil berakhir	Dijala
Urutan eksekusi	LIFO (Last In First Out)	Tidak
Pengaruh terhadap kode setelahnya	Kode setelahnya tetap dijalankan lebih dulu	Kode

Aspek	Defer	
Hubungan keduanya	Tidak dijalankan apabila exit dipanggil lebih dulu (pada Go)	Dapat dijalankan

2.11 Dampak dan Praktik Terbaik (Best Practice)

Berdasarkan pembahasan di atas, terdapat beberapa praktik terbaik yang disarankan dalam penggunaan defer dan exit, yaitu:

- Sebaiknya menghindari pemanggilan `os.Exit()` di dalam kode pustaka (library) dan hanya menggunakannya pada fungsi `main()` atau titik masuk program (entry point).
- Melakukan pembersihan sumber daya secara manual terlebih dahulu apabila memang program harus segera dihentikan menggunakan `exit`, misalnya dengan menutup berkas secara eksplisit sebelum memanggil `os.Exit()`.
- Mengutamakan alur keluar fungsi secara normal (`return`) dibandingkan penghentian paksa, agar seluruh mekanisme defer dapat berjalan sebagaimana mestinya.
- Memahami perbedaan perilaku `exit` pada setiap bahasa pemrograman yang digunakan, karena tidak semua bahasa memperlakukan defer atau `finally` dengan cara yang sama ketika `exit` dipanggil.
- Melakukan pengujian (testing) terhadap jalur-jalur kesalahan program untuk memastikan sumber daya tetap dibersihkan dengan baik meskipun program berhenti secara tidak normal.

BAB III

PENUTUP

3.1 Kesimpulan

Berdasarkan pembahasan yang telah diuraikan, dapat disimpulkan bahwa defer merupakan mekanisme untuk menjadwalkan pemanggilan fungsi agar dijalankan setelah fungsi pemanggil selesai dieksekusi, dengan urutan eksekusi mengikuti prinsip Last In First Out (LIFO). Mekanisme ini sangat berguna untuk memastikan proses pembersihan sumber daya berjalan dengan konsisten dan dapat diandalkan, tanpa harus menuliskannya secara berulang di setiap kemungkinan jalur keluar fungsi.

Adapun exit merupakan mekanisme untuk menghentikan eksekusi program secara langsung dan segera, disertai dengan pengembalian kode status kepada sistem operasi. Pada beberapa bahasa pemrograman seperti Go, pemanggilan `os.Exit()` tidak akan menjalankan defer yang telah dijadwalkan sebelumnya, sehingga penggunaannya perlu dilakukan dengan penuh kehati-hatian agar tidak menimbulkan kebocoran sumber daya maupun kesalahan logika pada program.

Dengan memahami perbedaan mendasar antara defer dan exit, seorang programmer diharapkan dapat merancang program yang lebih andal, aman, serta mudah dipelihara, khususnya dalam hal pengelolaan sumber daya dan penanganan kondisi kesalahan.

3.2 Saran

Penulis menyarankan agar pembaca, khususnya yang sedang mempelajari bahasa pemrograman Go maupun bahasa pemrograman lain yang memiliki mekanisme serupa, untuk selalu memperhatikan interaksi antara defer dan exit sebelum menerapkannya pada program yang

mengelola sumber daya penting. Selain itu, disarankan pula untuk memperbanyak praktik langsung (hands-on) dalam menulis dan menguji kode program agar pemahaman terhadap kedua konsep ini menjadi lebih mendalam dan aplikatif.

SOAL LATIHAN

Untuk mengukur pemahaman pembaca terhadap materi yang telah diuraikan dalam makalah ini, berikut disajikan beberapa soal latihan:

1. Jelaskan pengertian defer dalam pemrograman dan sebutkan bahasa pemrograman yang mempopulerkan konsep tersebut!
2. Jelaskan mengapa urutan eksekusi pernyataan defer mengikuti prinsip Last In First Out (LIFO), dan berikan contoh sederhananya!
3. Jelaskan pengertian exit dalam pemrograman beserta fungsi dari exit code!
4. Jelaskan apa yang terjadi pada pernyataan defer yang telah dijadwalkan apabila fungsi `os.Exit()` dipanggil pada bahasa pemrograman Go!
5. Sebutkan minimal tiga praktik terbaik (best practice) dalam menggunakan defer dan exit agar program tetap aman dan bebas dari kebocoran sumber daya!

DAFTAR PUSTAKA

Donovan, A. A. A., & Kernighan, B. W. (2016). The Go Programming Language. Addison-Wesley Professional.

Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.

The Go Authors. (2024). Effective Go. Diakses dari https://go.dev/doc/effective_go

The Go Authors. (2024). Package os Documentation. Diakses dari <https://pkg.go.dev/os>

Python Software Foundation. (2024). sys — System-specific parameters and functions. Diakses dari <https://docs.python.org/3/library/sys.html>

Oracle Corporation. (2024). Java Platform, Standard Edition Documentation. Diakses dari <https://docs.oracle.com/javase/>

LAMPIRAN

Lampiran 1. Contoh Program Ilustrasi Defer (Prinsip LIFO)

```
package main

import "fmt"

func tutup(nama string) {
    fmt.Println("Menutup:", nama)
}

func proses() {
    defer tutup("A")
    defer tutup("B")
    defer tutup("C")
    fmt.Println("Proses utama berjalan")
}

func main() {
    proses()
}

// Keluaran program:
// Proses utama berjalan
// Menutup: C
// Menutup: B
// Menutup: A
```

Lampiran 2. Contoh Program Ilustrasi Exit yang Melewati Defer

```
package main

import (
    "fmt"
    "os"
)

func proses() {
    defer fmt.Println("Baris ini TIDAK akan dijalankan")
}
```

```
        fmt.Println("Proses dimulai")
        os.Exit(1)
    }

    func main() {
        proses()
    }

    // Keluaran program:
    // Proses dimulai
    // (program berhenti dengan kode 1, defer tidak dijalankan)
```