1.

Ein Microkernel beschränkt sich auf grundlegende Speicherverwaltung und Kommunicaktion zwischen Prozessen.

Falsch: Der Microkernel kontrolliert auch Threads and Scheduling.

Im Kernelmodus muss das Betriebssystem darauf achten, dass die Speicherbereiche verschiedener Prozesse voneinander isoliert sind.

Wahr: Das IPC ist komplizierter bei Microkerneln, weil manche Prozesse in User Mode Komponenten gespeichert sind, und andere in Kernel Mode.

Interrupts können nicht unterbrochen werden.

Wahr: Hat Interrupt A höherer Priorität als Interrupt B, dann kannn A B unterbrechen.

Bei Microkerneln kommt es häufiger zu einem Kontextwechsel als bei monolithischen Kerneln.

Falsch: Es gilt genau das Gegentail. Die Schritte um eine Datei zu erstellen sind beispielsweise im Allgemeinenen bei einem Monolithische Kerneln folgende:
System Call(Interrupt) => Hardware => Syscall Handler =>File System. Bei einem Microkernel aber folgende: Application => Minimal Kernel => File System => Minimal Kernel => Device Driver => Hardware => .... => File System.
Gibt es 2 Kontextwechseln bei monolithischen Kerneln, dann treten mehr als 2 bei Microkerneln auf.

Die erste Implementierung von Syscalls (siehe Folie 2.21) is für ein Microkernel geeignet.

Falsch: Nutzerprozesse werden mit allen Rechten auf die Hardware ausgeführt, also gibt es keine Unterscheidung zwischen Nutzer und Kernel Modus, das geht auf einem Microkernel nicht.

Die zweite Implementierung von Syscalls (siehe Folie 2.22) ist für einen monolithischen Kernel geeignet.

Wahr: Die Serviceroutine kann ohne einen Prozesswechsel gebranched werden und operiert im selben Addressraum, das ist nur bei monolithischen Kerneln möglich.

In der dritten Implementierung von Syscalls (siehe Folie 2.23) werden Syscalls durch einen Nachrichtenaustausch realisiert.

Falsch: Der Syscall wird durch eine Calling Convention realisiert, denn der Microkernel reagiert auf einen call und regelt den Jump in eine Subroutine und den Return aus dieser.

In der vierten Implementierung von Syscalls (siehe Folie 2.24) werden Syscalls durch einen Nachrichtenaustausch realisiert.

Wahr: Denn der Microkernel sendet die Daten an einen Systemprocess, welcher die Daten empfängt, verarbeitet und wieder zurück sendet.

2.a)

```
/*
 *             TI3 Uebung2 Aufgabe2
 *      Authors: Tobias Lohse, Sven Klaus, Luisa Castano Jurado
 */

#include <stdio.h>

int valueOf(char letter);
char getCharByValue(int num);
char encodeChar(char char1, char char2);
char decodeChar(char char1, char char2);
void encode(void);
void decode(void);
void readPass();

char userInput[10] = { 0 };
char pw[10] = { 0 };
char longPw[100] = { 0 };
char clearTxt[100] = { 0 };
char codeTxt[100] = { 0 };
char chiffreTxt[100] = { 0 };
char decodeTxt[100] = { 0 };
int idx = 0;

int main(int argc, char const *argv[]) {

	//menu
	while (1) {
		printf("(D)ecode or (E)ncode Mode? \n");
		fgets(userInput, sizeof(userInput), stdin);

		if (userInput[1] == '\n' && userInput[0] == 'd') {
			decode();
			break;
		} else if (userInput[1] == '\n' && userInput[0] == 'e') {
			encode();
			break;
		} else {
			printf("Please enter 'd' or 'e' to continue.");
		}
	}
	return 0;
}

void decode(void) {
	printf("DECODE:\n");
	readPass();

	printf("Enter the encoded Text: \n");
	fgets(chiffreTxt, 99, stdin);

	// repeat pass for length of text
	idx = 0;
	for (int i = 0; i < 100; ++i) {
		if (chiffreTxt[i] && (chiffreTxt[i] != 10)) {
			do {
				if (pw[idx % 11]) {
					longPw[i] = pw[idx % 11];
				}
				idx++;
			} while (!pw[idx % 11]);
		}
	}

	// decode every char
```

```
	for (int i = 0; i < 100; ++i) {
		if (chiffreTxt[i] && (chiffreTxt[i] != 10)) {
			decodeTxt[i] = decodeChar(chiffreTxt[i], longPw[i]);
		}
	}

	printf("\n");
	printf("Passwort  : %s\n", pw);
	printf("Codewort  : %s\n", longPw);
	printf("Chiffre   : %s\n", chiffreTxt);
	printf("-------------------------------------------\n");
	printf("Decode    : %s\n", decodeTxt);
	printf("-------------------------------------------\n");

}

void encode(void) {
	printf("ENCODE:\n");
	readPass();

	printf("Enter the Text: \n");
	fgets(clearTxt, 99, stdin);

	// remove unused signs & spaces - convert to UpperCase
	idx = 0;
	for (int i = 0; i < 100; ++i) {
		char c = clearTxt[i];
		if (valueOf(c) >= 0) {
			codeTxt[idx] = getCharByValue(valueOf(c));
			idx++;
		}
	}

	// repeat pass for length of text
	idx = 0;
	for (int i = 0; i < 100; ++i) {
		if (codeTxt[i]) {
			do {
				if (pw[idx % 11]) {
					longPw[i] = pw[idx % 11];
				}
				idx++;
			} while (!pw[idx % 11]);
		}
	}

	//encode every char
	for (int i = 0; i < 100; ++i) {
		if (codeTxt[i]) {
			chiffreTxt[i] = encodeChar(codeTxt[i], longPw[i]);
		}
	}

	printf("\n");
	printf("Passwort  : %s\n", pw);
	printf("Klartext  : %s\n", clearTxt);
	printf("Codierung : %s\n", codeTxt);
	printf("Codewort  : %s\n", longPw);
	printf("-------------------------------------------\n");
	printf("Chiffre   : %s\n", chiffreTxt);
	printf("-------------------------------------------\n");
}

void readPass() {
	printf("Enter a Passphrase: \n");
	fgets(userInput, sizeof(userInput), stdin);

	idx = 0;
	for (int i = 0; i < 10; ++i) {
		if (userInput[i]) {
			if (getCharByValue(valueOf(userInput[i]))) {
				pw[idx] = getCharByValue(valueOf(userInput[i]));
				idx++;
			}
		}
	}
	printf("Acceptet Passphrase: %s\n", pw);
}
```

```
char encodeChar(char char1, char char2) {
        int charSum = valueOf(char1) + valueOf(char2);
        return getCharByValue(charSum % 36);
}

char decodeChar(char char1, char char2) {
        int charDiff = valueOf(char1) - valueOf(char2) + 36;
        return getCharByValue(charDiff % 36);
}

int valueOf(char letter) {
        int num = (int) letter;
        // Buchstaben A-Z
        if (num >= 65 && num <= 90) {
                return num - 65;
                // Buchstaben a-z
        } else if (num >= 97 && num <= 122) {
                return num - 97;
                //Zahlen 0-9
        } else if (num >= 48 && num <= 57) {
                return num - 22;
        } else {
                return -1;
        }
}

char getCharByValue(int num) {
        // Buchstaben A-Z
        if (num >= 0 && num <= 25) {
                return (char) num + 65;
                // Zahlen 0-9
        } else if (num >= 26 && num <= 35) {
                return (char) num + 22;
        } else {
                return (char) 0;
        }
}
```

2.b)

Wenn das Codewort, im Gegensatz zum Text, relativ kurz ist, kann die Länge des Codeworts relativ leicht ermittelt werden. Bei der Codierung von nur einem Buchstaben läßt sich die Länge des Codeworts schon mit bloßem Auge erkennen:

```
Passwort   : 1234
Klartext   : eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee

Codierung  : EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
Codewort   : 1234123412341234123412341234123412341234123412
----------------------------------------------------------
Chiffre    : 5678567856785678567856785678567856785678567856
----------------------------------------------------------
```

In einem normalen Text wird es, auch nach der Vigenere-Chiffre Codierung, Stellen geben, die sich periodisch wiederholen. Deswegen läßt sich, ab einer gewissen Länge des Textes, mithilfe des Kasiski-, oder Friedman-Tests, die Länge des Codeworts ermitteln. Ist die Länge bekannt kann man den Code, wie bei einer typischen Caesar-Verschlüsselung, mittels Häufigkeitsanalyse knacken.

Quellen:
http://de.wikipedia.org/w/index.php?title=Polyalphabetische_Substitution
http://www.iti.fh-flensburg.de/lang/krypto/klassisch.htm